Quando um problema não tiver solução, inventamos uma...

18 de março de 2021

Quando encontramos um problema que não sabemos resolver a primeira ideia é encontrar uma solução pré-existente em algum lugar. Isso é particularmente verdade em problemas matemáticos, para os quais muita coisa já está desenvolvida e é fácil encontrar a resposta que procuramos. A internet, livros, referências, artigos etc estão aí para essa finalidade. Mas e quando não encontramos a solução em lugar nenhum? Mesmo se o problema for considerado muito simples?

Trago duas abordagens possíveis: desistir ou inventar uma solução. Devo lembrar que em um cenário parecido com o atual, no caso uma pandemia de peste negra, Newton inventou técnicas até então desconhecidas para resolver os problemas que o interessavam. Vou apresentar aqui um problema aparentemente simples, de solução ainda mais óbvia, porém que não encontrei a solução em lugar algum. Como não poderia desistir dele, fui forçado a inventar uma solução para o mesmo. Pode ser que essa solução exista e esteja publicada, claro, mas não a encontrei e, por isso, na limitação do meu conhecimento, a considerarei como original.

1) O problema de minimização do traço de um produto de matrizes

O problema em questão surgiu num contexto de quantificação de emaranhamento. A grande área na qual esse problema se insere é bastante complicada e quase todos os problemas são dificílimos de resolver. O que me surgiu foi a seguinte questão, colocada em termos matemáticos a seguir. Considere duas matrizes A e B, ambas quadradas de dimensões $n \times n$. Queremos encontrar a solução para a seguinte otimização:

$$\min Tr(AB) \tag{1}$$
$$\text{s.a. } Tr(A) = 1,\ A \geq 0, \tag{1}$$

em que B é uma matriz conhecida, que satisfaz a propriedade: $B = B^{\dagger}$.

Pra quem está sem entender nada, uma breve explicação: matrizes, como A e B que mencionei, são simplesmente blocos de números. No caso temos duas matrizes que eu disse serem quadradas, ou seja, o número de linhas é igual ao número de colunas. Então temos dois blocos de números, A e B , que são quadrados (pra quem conhece Excel, uma planilha com vários dados nas células é uma representação de matrizes). Matrizes são de imensa importância em matemática, engenharia, física, computação, estatística etc. Sabe aquele exame de ressonância magnética? A trajetória da granada num jogo de tiro? Pois é, pode ter certeza que em tudo isso tem matrizes.

E o que seria o resto? Sabemos quem são A e B, mas e o resto? Bom, Tr simboliza uma operação de matrizes, chamada de **traço**. Lembra do número de colunas e linhas? Pois é, os elementos da matriz cujo número de linhas é igual ao número de colunas são chamados de diagonal principal. O traço é simplesmente a soma dos elementos da diagonal principal de uma matriz.

O leitor, embora leigo no assunto, porém atento ao texto, irá se questionar: mas, se o traço é uma operação sobre matrizes, como ele atua no produto AB? Simples: o produto de duas matrizes também é uma matriz! Logo, o problema que encontrei consiste em minimizar (ou seja, encontrar o menor valor) do traço do produto AB! Todo essa frase pode ser expressa matematicamente por $\min Tr(AB)$, eis o poder de síntese da notação matemática...

Esses problemas são chamados de **problemas de minimização** e pertencem a uma classe mais geral, chamados de problemas de otimização. Os problemas de minimização consistem em encontrar o menor valor de uma dada função (chamada de função objetivo) dado um conjunto de restrições. Alguns problemas podem ser com poucas restrições, outros com inúmeras restrições etc. No nosso caso a função objetivo é $Tr(AB)$ e as restrições (abreviadas por s.a., sujeito a) são: o traço de A tem que ser igual a 1 ($Tr(A) = 1$) e a matriz A tem que ser positiva semidefinida ($A \geq 0$). Sobre essa última restrição, A ser positiva semidefinida, não comentaremos muito aqui, é importante colocarmos que isso é uma restrição do problema, que faz todo sentido no contexto onde ele surgiu. Essa definição é um pouco complexa para um texto curto como esse então deixo aqui uma referência sobre o assunto [1].

Por fim, falta analisarmos B. Como se trata de um problema de minimização queremos encontrar A que minimiza a função objetivo $Tr(AB)$. Porém, para que isso funcione, obviamente, B tem que ser conhecida! E, de fato, B é conhecida e possui a propriedade de ser Hermitiana ($B = B^\dagger$). E o que significa isso? As matrizes A e B podem ser tanto reais (seus elementos são números reais) ou quanto complexas (seus elementos são números complexas). Além disso, existe uma operação em matrizes, chamada de **transposta** que troca linhas por colunas. Na operação de transposição, trocamos os elementos das linhas pelo das colunas e vice-versa. A operação $B^\dagger$ mencionada acima é a combinação de uma transposição da matriz, seguida de uma conjugação complexa (para quem conhece, trocar o elemento imaginário i por $-i$). Quando uma data matriz B é igual à sua conjugada transposta $B^\dagger$, dizemos que a mesma é Hermitiana. De novo, mesmo que você não entenda o que essas coisas significam, é importante saber que essa é uma propriedade de B que também faz todo o sentido no contexto em que o problema está inserido. Mais sobre matrizes Hermitianas e suas propriedades em [1].

Pronto, agora sim temos um entendimento básico do que cada coisa significa. Relembrando: queremos minimizar o traço do produto AB para uma matriz B conhecida e Hermitiana, sujeito às restrições de que A é de traço 1 e positiva semidefinida. Esse é o problema. Antes de passarmos para a solução e suas dificuldades, comentaremos primeiro sobre um exemplo clássico de otimização, muito instrutivo no meu entender: o problema do caixeiro-viajante.

2) O problema do caixeiro-viajante

Imagine que você é um vendedor, atuante na área comercial de uma empresa que comercializa produtos. Um dos seus gerentes lhe entrega uma lista de destinos, quatro no total, São Paulo, Belo Horizonte, Rio de Janeiro e Brasília. E lhe dá a seguinte tarefa: você deverá partir de São Paulo, cidade-sede da empresa, e deverá visitar todas as outras, sem repetir nenhuma delas, retornando em seguida para a cidade de origem, São Paulo. Até aí, tudo ótimo. Você está feliz com a tarefa que lhe fora dada: serão quatro visitas, sem repetir cidades e depois retornar para casa.

Mas eis que seu chefe tira do bolso a condição mais frustrante: essas visitas devem ser feitas realizando **o menor caminho possível**. Essa decisão está embasada em reduzir os custos de combustível e minimizar tempo gasto com viagens.

Vejamos, então, as distâncias (via Google) entre essas cidades:

Cidades	Belo Horizonte	Brasília	Rio de Janeiro	São Paulo
Belo Horizonte	0	624 km	339 km	489 km
Brasília	624 km	0	933 km	873 km
Rio de Janeiro	339 km	933 km	0	357 km
São Paulo	489 km	873 km	357 km	0

Note que essa tabela também é uma matriz (os números dentro dela, desconsiderando os nomes das cidades) que ainda por cima tem traço nulo e é simétrica (os elementos acima da diagonal principal são iguais aos elementos abaixo da diagonal principal).

Com os dados em mãos, somos capazes de resolver o problema proposto pelo gerente da seguinte forma: listamos todas as possibilidades de trajetos e calculamos a distância total percorrida em cada trajeto. O trajeto ótimo (a solução do problema), também chamado de trajeto mínimo ou trajeto de menor distância, é aquele que terá a menor distância total percorrida. Encontrando esse trajeto resolvemos o problema. Pra simplificar chamamos as cidades Belo Horizonte, Brasília, Rio de Janeiro e São Paulo de 1, 2, 3 e 4, respectivamente. Listamos a seguir todas as possibilidades de trajeto e suas respectivas distâncias percorridas:

Trajeto	Distância total percorrida
4-1-2-3-4	2403 km
4-1-3-2-4	2634 km
4-2-1-3-4	2193 km
4-2-3-1-4	2634 km
4-3-1-2-4	2193 km
4-3-2-1-4	2403 km

Note que devido às restrições do problema (partir de São Paulo e retornar para essa mesma cidade) as soluções que são simétricas, como é o caso da primeira com a última, da segunda com a quarta e da terceira com a quinta. Como queremos minimizar a distância percorrida, as duas soluções que igualmente minimizam o problema são: 4-2-1-3-4 e 4-3-1-2-4 que correspondem às viagens, respectivamente: São Paulo - Brasília - Belo Horizonte - Rio de Janeiro - São Paulo e São Paulo - Rio de Janeiro - Belo Horizonte - Brasília - São Paulo. Ambos trajetos fornecerão a menor distância possível.

Imagine agora que seu chefe tenha dado um outro problema: você deve partir de São Paulo e viajar toads as demais cidade uma única vez cada, sem repetir nenhuma delas, percorrendo a menor

distância possível, **sem a necessidade de retornar a São Paulo**. Nesse caso a tabela de distância total percorrida para todos os possíveis trajetos ficará ligeiramente diferente:

Trajeto	Distância total percorrida
4-1-2-3	2046 km
4-1-3-2	1761 km
4-2-1-3	1836 km
4-2-3-1	2145 km
4-3-1-2	1320 km
4-3-2-1	1914 km

Note que agora as distâncias não estão mais aos pares e a solução ótima é única: o trajeto 4-3-1-2 que corresponde ao caminho São Paulo - Rio de Janeiro - Belo Horizonte - Brasília, percorrendo a menor distância possível.

Mas o que esses dois problemas diferentes tem em comum? Ambos envolvem a passagem por cidades apenas uma única vez, sem repetir nenhuma cidade, percorrendo a menor distância possível. Problemas dessa natureza são chamados de **caixeiro-viajante**. Essa é uma classe de problemas relativamente antiga e, pode parecer incrível, muito difícil de resolver. O problema do caixeiro-viajante (travelling salesman problem) surgiu no contexto de viagens de mercadores, exatamente como posto no início dessa seção, mas hoje evoluiu para outros contextos como teoria de grafos, logística e distribuição pra citar alguns. Ele pertence a uma classe de problemas chamados NP-difíceis, que significa, grosso modo, problemas absurdamente demorados e difíceis de se resolver, mesmo com os melhores computadores do mundo.

E de onde vem tanta dificuldade? Com um total de $N = 4$ cidades, tivemos $N = (4-1)! = 3! = 3 \times 2 \times 1 = 6$ possibilidades de trajetos diferentes. Para $N = 20$, por exemplo, temos $19! = 19 \times 18 \times 17 \ldots \times 1 = 121645100408832000$ possibilidades de trajetos! E, pior ainda: quanto mais aumentamos N, mais rápido ainda o problema se torna mais difícil. Um exemplo extremo: resolver o problema do caixeiro-viajante para $N = 101$ leva 100 vezes mais tempo do que resolver o problema para $N = 100$.

Com isso fica claro que encontrar estratégias (ou algoritmos, como queira) que resolvam problemas dessa dificuldade de forma eficiente é extremamente importante. O problema original que temos, minimizar o traço de uma matriz produto de outras duas, não é nem de longe tão complexo quanto o problema do caixeiro-viajante, mas apresenta um certo grau de complexidade.

Note também o quanto a matemática é mais próxima de nós do que podemos imaginar: um problema tão "inocente" quanto a trajetória de um viajante avarento, disposto a gastar apenas o mínimo com deslocamento, é extremamente difícil do ponto de vista computacional. Existem inúmeros outros problemas de interesse prático que são tão difíceis (ou até piores) do que esse. E, claro, o próprio problema do caixeiro-viajante original pode se tornar ainda pior, incluindo mais restrições, como por exemplo uma certa cidade X não pode ser visitada logo após uma outra cidade Y, o trajeto entre algunas cidades ser impossível e por aí vai. A área de logística é recheada de

problemas desse estilo. Portanto, reforçamos, saber resolver problemas da melhor forma possível (ou, ao menos, resolvê-los de forma *eficiente*) é essencial em modelagem matemática. Para mais detalhes sobre o problema do caixeiro-viajante, suas variações e soluções recomendamos [2].

OBS: Uma observação adicional final é que todas as distâncias computadas na tabela são em viagens aéreas (por simplicidade). O problema é igualmente bem posto se as distâncias forem as rodoviárias, diferindo apenas nas solução final e na distância total percorrida. Usamos as distâncias aéreas apenas como uma questão de facilidade para obter os dados.

3) Resoluções por força bruta

A primeira abordagem em um problema de otimização (sendo amplo, qualquer problema) é resolvê-lo por força bruta. No contexto que estamos tratando essa ideia consiste em: elenca-se todas as possíveis soluções para o problema e calculamos a função objetivo a ser minimizada (ou maximizada, o procedimento é o mesmo) para cada uma dessas possibilidades. Escolhemos, em seguida, a solução que corresponde ao menor valor da função objetivo. Essa é, em linhas gerais e bem simples, a abordagem de resolução via força bruta.

Pense um pouco a esse respeito. Encontrar não um, não dois, mas **todas** as possíveis soluções de um dado problema de otimização, avaliar o problema nessas soluções (computando a função objetivo, seja qual ela for, de algum modo) e encontrar a solução que otimiza o problema. Em alguns casos, isso não só é possível de ser feito como garante o melhor resultado: testando todas as soluções, é garantido encontrar a menor delas. Em outros casos, isso é virtualmente impossível de ser feito, seja porque o espaço de soluções é virtualmente infinito ou seja porque esse método levaria tempo demais para obter a solução desejada.

A solução que mostramos na seção anterior para o problema do caixeiro-viajante é baseada nessa ideia de resolução por força bruta. Note que enquanto mantivermos o número de cidades pequeno, N da ordem de 8, conseguimos mapear todas as soluções (para $N = 8$ são 40320 soluções) mas a partir disso o número de soluções é grande demais para que possamos avaliar todas elas. Daí surgem diversos outros métodos: gradiente descendente, linearização, algoritmos genéticos, algoritmos bioinspirados, programação linear, métodos simpléticos e por aí vai. Dependendo do problema a ser estudado, um método será melhor do que o outro.

4) Solução do problema de minimização do traço de um produto de matrizes

Depois dessa longa introdução e exposição sobre problema de otimização de um modo geral, além da apresentação de um problema tipicamente difícil nessa área, retornemos ao nosso problema original da equação (1). Para relembrar: queremos encontrar A que minimize o $Tr(AB)$ sujeito às restrições e sendo que B é conhecida e com propriedades conhecidas.

Uma primeira consideração é sobre o espaço de soluções: matrizes são, como falamos, blocos de números e, no nosso caso, são matrizes quadradas $n \times n$. Se todos seus elementos forem diferentes, isto é nenhuma informação sobre seus elementos for fornecida, temos um total de n^2 elementos diferentes em A. Como esses elementos podem ser reais ou complexos, o espaço das

possíveis soluções é teoricamente infinito! Nesse caso, seria impossível encontrar um algoritmo de força bruta que calculasse todas as soluções possíveis, pois é impossível encontrar um número infinito de soluções. Porém, isso não impede que métodos de força bruta sejam usados e veremos isso mais à frente como fazê-lo.

Além disso, uma propriedade importante de B tem que ser levada em conta: sua semidefinição positiva ou negativa. Trataremos os dois casos separadamente a seguir.

4.1) B é positiva semidefinida

Se B é uma matriz positiva semidefinida e como A tem como restrição ser positiva semidefinida, teremos um produto AB de duas matrizes positivas semidefinidas, o que também é positiva semidefinida. E o traço de matrizes positiva semidefinidas é sempre positivo, logo o menor valor possível para $Tr(AB)$ é zero]. Dependendo de onde esse problema está inserido, essa pode ser uma solução aceitável: o menor valor pro traço vai ser nulo. Vejamos o próximo caso.

4.2) B é negativa semidefinida

Como B, por definição, é hermitiana (seu conjugado transposto é igual a si mesma) ela terá todos os seus autovalores reais (uma propriedade de matrizes hermitianas, vide [1]). Além disso ela também é diagonalizável. E o que significa ser diagonalizável? Significa que é possível realizar algumas operações nessa matriz para se obter uma matriz diagonal, cujos únicos elementos não-nulos estão na diagonal principal. Essa matriz diagonal é obtida a partir de um procedimento de diagonalização de matrizes e as entradas dessa diagonal são o que chamamos de **autovalores** da matriz. Diagonalização de matrizes e autovalores são parte importantíssima de toda ciência moderna (e não, isso não foi uma hipérbole).

Como B é diagonalizável podemos calcular todos os seus autovalores e se ela for negativa semidefinida, pelo menos um de seus autovalores será negativo. Associado a cada autovalor há um conjunto de números chamado de autovetor. Escolhemos os autovetores associado aos autovalores negativos de B e construímos um projetor a partir deles. E ele servirá para projetar no espaço de autovalores negativos de B. Por fim escolhemos a solução ótima A^* como sendo esse projetor P.

Colocando em termos mais matemáticos considere que B possua m autovalores negativos, com $m < n$, sendo n a dimensão de B. Pra cada autovalor λ_m existe um autovetor associado v_m tal que satisfaz a relação de autovalores:

$$Bv_n = \lambda_n v_n. \tag{2}$$

Construímos o projetor $p_m = v_m v_m^\dagger$, em que † denota o conjugado transposto, para um autovetor em particular. Somando sobre todos os autovetores associados a autovalores negativos:

$$P = \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} v_i v_i^\dagger, \tag{3}$$

e, por fim, escolhemos a solução ótima como sendo $A^* = P$. Note que como p_i é projetor e está escrito na forma $v_i v_i^\dagger$, o mesmo será positivo semidefinido automaticamente. Além disso como os autovetores são todos normalizados e p é projetor, temos que $Tr(p)$. Por fim, a soma de projetores

também é um projetor, logo P é de traço 1 e como a soma de matrizes positivas semidefinidas é também positiva semidefinida, o projetor P será positivo semidefinido. Com isso encontramos a solução ótima $A^* = P$ com as restrições corretamente satisfeitas.

5) Alguns testes e benchmarks

Na seção anterior provamos que a escolha de $A^* = P$, sendo P o projetor dos autovalores negativos de B garante a minimização do problema. Agora iremos testar essa ideia usando recursos computacionais (leia-se: Python). Começaremos com um caso mais simples. Considere B uma matriz 2×2 dada por:

$$B = \begin{pmatrix} -1/2 & 1/2 \\ 1/2 & -1/2 \end{pmatrix}, \tag{4}$$

essa matriz tem autovalores $\lambda_1 = -1$ e $\lambda_2 = 0$ e os autovetores associados são: $v_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ e $v_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. O projetor dos autovetores negativos será:

$$P = v_1 v_1^\dagger = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{pmatrix} = A^*. \tag{5}$$

Note que A^* satisfaz, automaticamente: $Tr(A^*) = 1$ e seus autovalores são: 1 e 0. Logo, como seus autovalores são todos positivos ou iguais a zero, a matriz A^* é positiva semidefinida. Ou seja, acabamos de confirmar, comum exemplo prático, o que provamos na seção anterior: A^*, de fato, satisfaz as restrições pro problema e o valor mínimo será:

$$\min Tr(AB) = Tr(A^* B) = -1. \tag{6}$$

Nesse exemplo mínimo é razoável usar algoritmos de força bruta. Conforme comentamos antes, o espaço de soluções é teoricamente infinito, pois existem infinitas matrizes de entradas reais de dimensão 2×2. Porém, é possível realizar um algoritmo de força bruta aleatório, que consiste em sortear um número grande de soluções aleatórias satisfazendo uma dada distribuição de probabilidade (usualmente distribuição de probabilidade uniforme) e computar o valor $Tr(AB)$ pra todas essas soluções e, em seguida, selecionar a solução que garante o menor valor pra essa função. Com 5 milhões de sorteios aleatórios encontramos: $\min Tr(A_{bf} B) = -0.994$, em que A_{bf} denota a solução via força bruta. Importante notar que o método via autovalores é muito mais rápido que o método de força bruta aleatório. E, claro, esse último é impraticável para matrizes grandes.

Mas não basta resolver o problema para matrizes pequenas 2×2. Por isso, a seguir, apresentamos um código mínimo em Python que resolve o problema para matrizes 1000×1000 e computaremos o tempo gasto nesse processo.

In [150…

```
import numpy as np
import time

time_start = time.time()
dim = 1000
```

```
C = np.array([[-0.5,0.5],[0.5,-0.5]]) #Minimal problem, showed above in the text
B = np.random.rand(dim,dim) #Construct a random B to calculate Tr(AB)
B = B*B.conj().T #Ensure B is Hermitian
B = B/np.trace(B) #Ensure trace(B) = 1

#Diagonalize B
d, u = np.linalg.eig(B) #d is the vector of eigenvalues and u is the eigenvector matrix
size = len(d) #Maximum number of eigenvalues of B
P = 0.0 #Projector
#Construct the projector for negative eigenspace of B
for i in range(size):
    if d[i] < 0.0:
        vec = np.array([u.T[i]]) #Important to transpose to pick the correct eigenvecto
        P = P + np.kron(vec,vec.T)

time_elapsed = (time.time() - time_start)

#Compute the minimum value and print the optimal solution
print('The minimial value is:')
print(np.trace(np.dot(P,B)))
print('With optimal solution:')
print(P)
print('And the time consumed in this calculation was:')
print(time_elapsed, 'seconds')
```

```
The minimial value is:
-8.959982906665438
With optimal solution:
[[ 0.47510741 -0.00425454  0.00064783 ...  0.02206769  0.00763345
   0.00202507]
 [-0.00425454  0.5164813   0.0087008  ...  0.01215705  0.00734246
   0.0232711 ]
 [ 0.00064783  0.0087008   0.51087447 ... -0.0109346   0.00554144
  -0.03170756]
 ...
 [ 0.02206769  0.01215705 -0.0109346  ...  0.51317942  0.00938147
  -0.00302559]
 [ 0.00763345  0.00734246  0.00554144 ...  0.00938147  0.48460595
   0.01845752]
 [ 0.00202507  0.0232711  -0.03170756 ... -0.00302559  0.01845752
   0.47907025]]
And the time consumed in this calculation was:
11.167887687683105 seconds
```

In [171…

```
import numpy as np
import time

time_start = time.time()
C = np.array([[-0.5,0.5],[0.5,-0.5]])
tr = [0.0]
allA = [0.0]

for i in range(5000000):
    A = np.random.rand(2,2) #Construct a random A
    A = A*A.conj().T #Ensure A is Hermitian
    Atrace = np.trace(A)
    A = A/np.trace(A) #Ensure trace(A) = 1
    tr.append(np.trace(np.dot(A,C))*Atrace) #Save all objective function evaluations
    allA.append(A) #Save all A

time_elapsed = (time.time() - time_start)
index = tr.index(min(tr))
```

```
print('The minimal value is:')
print(min(tr))
print('With optimal solution:')
print(allA[index])
print('And the time consumed in this calculation was:')
print(time_elapsed, 'seconds')
```

```
The minimal value is:
-0.9959065805878847
With optimal solution:
[[0.50000073 0.00128977]
 [0.00128977 0.49999927]]
And the time consumed in this calculation was:
146.66924834251404 seconds
```

Como os cálculos anteriores mostraram, o tempo gasto pra se resolver o problema com uma matriz 1000×1000, via método dos autovalores, é bem menor do que o tempo gasto para se resolver o mesmo problema só que com uma matriz 2×2, via método de força bruta.

Outro comentário é que, aparentemente, o método de força bruta gerou resultado diferente daquele obtido pelo método dos autovalores, para o caso exemplo da matriz 2×2, equação (4). Porém ambos resultados são equivalentes. As matrizes obtidas por ambos métodos tem traço 1 e são positivas semidefinidas, portanto são ambas soluções aceitáveis para o problema. A única diferença significativa é que a solução via método de força bruta não é um projetor enquanto que a matriz obtida pelo método dos autovalores o é. Isso acontece porque o espaço de soluções (virtualmente infinito!) é grande demais para que um método de força bruta convirja pro resultado exato com apenas 5 milhões de repetições. Uma forma de contornar isso é restringir o espaço de busca das soluções aceitáveis pro método de força bruta para as matrizes que também são projetores, isto é, matrizes de traço 1. O código a seguir realiza essa implementação.

In [223…

```
import numpy as np
import time

time_start = time.time()
C = np.array([[-0.5,0.5],[0.5,-0.5]])
tr = [0.0]
allA = [0.0]

for i in range(5000000):
    A = np.array([np.random.rand(2)]) #Construct a random vector A
    A = np.kron(A,A.T) #Ensure A is an projector
    Atrace = np.trace(A)
    A = A/np.trace(A) #Ensure trace(A) = 1
    tr.append(np.trace(np.dot(A,C))*2) #Save all objective function evaluations
    allA.append(A) #Save all A

time_elapsed = (time.time() - time_start)
index = tr.index(min(tr))
print('The minimal value is:')
print(min(tr))
print('With optimal solution:')
print(allA[index])
print('And the time consumed in this calculation was:')
print(time_elapsed, 'seconds')
```

```
The minimal value is:
-0.9999996657674326
With optimal solution:
```

```
[[2.79278523e-14 1.67116284e-07]
 [1.67116284e-07 1.00000000e+00]]
And the time consumed in this calculation was:
365.9011917114258 seconds
```

Note que, agora sim, obtivemos um projetor como solução, a matriz $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$, que não é senão a representação diagonal da solução ótima A^* encontrado pelo método de autovalores. Em outras palavras: ambas soluções são análogas pois pode-se mapear uma através da outra via uma diagonalização. A grande diferença, claramente, são os 0,001 segundos de execução para o método de autovalores contra 365 segundos do método força bruta. Repare que o conhecimento prévio do problema (quais soluções esperar, onde procurar etc) é muito valioso para se conseguir resolvê-lo. Pra encerrar, consideremos a solução via força bruta para o problema de otimização de uma matriz B positiva semidefinida. Queremos confirmar que o resultado da minimização é zero. Vem comigo!

In [226…

```
import numpy as np
import time

time_start = time.time()
C = np.array([[0.5,0.5],[0.5,0.5]])  #Now C is positive semidefinite with eigenvalues 1
tr = [0.0]
allA = [0.0]

for i in range(5000000):
    A = np.random.rand(2,2) #Construct a random A
    A = np.dot(A,A.conj().T) #Ensure A is Hermitian
    Atrace = np.trace(A)
    A = A/np.trace(A) #Ensure trace(A) = 1
    tr.append(np.trace(np.dot(A,C))*Atrace) #Save all objective function evaluations
    allA.append(A) #Save all A

time_elapsed = (time.time() - time_start)
index = tr.index(min(tr))
print('The minimal value is:')
print(min(tr))
print('With optimal solution:')
print(allA[index])
print('And the time consumed in this calculation was:')
print(time_elapsed, 'seconds')
```

```
The minimal value is:
0.0
With optimal solution:
0.0
And the time consumed in this calculation was:
142.57995700836182 seconds
```

6) Considerações finais

Conseguimos resolver um problema aparentemente simples com uma solução igualmente simples! Yes!

Sim, é muito provável que essa solução já exista em algum lugar. E, como eu não a achei, resolvi escrever esse texto e deixar registrado para caso alguém precise. E, aí está!

No processo para resolver o problema acrescentei comentários sobre soluções de problemas de otimização, apresentei o famoso problema do caixeiro-viajante e alguns comentários rápidos sobre

matrizes. Esses comentários, talvez tido como muito básicos ou introdutórios, foram pensados para qualquer pessoa ser capaz de entender ou, ao menos, de acompanhar a construção da solução.

E o mais importante aqui nem é a solução em si, ou o detalhamento técnico para obtê-la. É o **processo**. Em qualquer área quantitativa, seja matemática, física, engenharia, ciência de dados etc enfrentaremos inúmeros problemas semelhantes a esse: problemas estes que não sabemos resolver de início. E, baseados em nossos conhecimentos da área, temos que ser capazes de procurar as soluções nos lugares certos. E, se não a encontrarmos, inventamo-la! Trabalhar com modelagem matemática/quantitativa de problemas reais é um misto de técnica, arte, pesquisa e persistência.

Quaisquer dúvidas, críticas, comentários, sugestões, terei prazer em ler. Meu email: jvfrossard@gmail.com

Muito obrigado e até a próxima!

Referências

[1] ANTON, Howard; RORRES, Chris. Álgebra linear com aplicações. Porto Alegre: Bookman, 2001.

[2] APPLEGATE, David L. et al. The traveling salesman problem: a computational study. Princeton university press, 2006.