L'algorithme QR

L'algorithme QR permet de calculer une factorisation de Schur d'une matrice A

A = Q . T . Q^T

où Q est orthogonale et T est triangulaire supérieure.
Sur la diagonale de T, on trouve une approximation des valeurs propres de A.

Dans le cas général, les colonnes de Q ne sont pas les vecteurs propres.
Mais ce n'est pas un problème : avec la méthode de la puissance inverse, on peut calculer un vecteur propre par valeur propre en utilisant comme approximations les valeurs sur la diagonale de T. Et donc on peut diagonaliser A dans le cas où A est diagonalisable.

Dans le cas où A est une matrice symétrique,
Q contient les vecteurs propres de A (Q = X avec les notations du cours)
T est diagonale (T = Lambda avec les notations du cours).

Il est évident que T est diagonale parce que A = A^T (puisque A est symétrique) et donc :

A = A^T = (Q . T . Q^T)^T = Q . T^T . Q^T et donc T = T^T.
Et une matrice triangulaire supérieure égale à sa transposée est diagonale.

La factorisation de Schur est une diagonalisation de A.

Pourquoi c'est important en GIS ? C'est à cause de l'application à l'analyse en composantes principales (ACP).

Idée : si vous partez d'une matrice M rectangulaire, qui contient des données.
Pour faire une ACP en deux dimensions de M :

Méthode historique :
1. On calcule A = M^T . M et on obtient une matrice symétrique (valeurs propres réelles)
2. On cherche les deux valeurs propres les plus grandes en valeur absolue lambda1 et lambda2
3. On cherche deux vecteurs propres orthogonaux q1 et q2 de A : un pour chaque valeur propre lambda1 et lambda2.
4. Les composantes principales sont les vecteurs propres q1 et q2.
5. Il y a une technique simple qui permet de projeter les données de la matrice M sur le plan formé par q1 et q2.

Au passage, il y a une méthode moderne :
1. Au lieu de calculer les valeurs propres de M^T . M on calcule la décomposition en valeurs singulières de M. On évite de former la matrice A = M^T . M
2. C'est la même idée et on peut montrer que les valeurs propres de A sont les carrés

des valeurs singulières de M.

Ça s'appelle la SVD en Anglais (Singular Value Decomposition).

L'idée est très naturelle. Toute matrice (même rectangulaire) a une SVD. C'est une factorisation de matrices qui ressemble à ça :

M = U . Sigma . V^T où :

U et V sont orthogonales
Sigma est diagonale
Les éléments diagonaux de Sigma sont positifs ou nuls : ce sont les valeurs singulières de M.

Fin de la parenthèse :

On voit (méthode historique) que le calcul des valeurs propres des matrices symétriques a des applications en statistique.

On montre l'algorithme QR (surtout ne pas confondre avec la factorisation QR) sur un exemple.

Exo. On voudrait calculer une matrice symétrique ayant des valeurs propres fixées : -74, 38, 2, et 42.
Sauriez-vous faire ça ?

Il suffit de mettre les valeurs propres sur la diagonale de Lambda et de calculer

A = Q . Lambda . Q^T

On vérifie facilement que A^T = A : Puisque A^T = Q . Lambda^T . Q^T et Lambda^T = Lambda.
Les valeurs propres de A sont données par Lambda. Parce que, comme Q est orthogonale, Q^T = Q^(-1) et donc Lambda et A sont semblables.

Pour calculer une matrice orthogonale Q aléatoire, il suffit de calculer la factorisation QR d'une matrice aléatoire.

```
> with(LinearAlgebra):
```

Une matrice « aléatoire »

```
> A0 := <<1,1,1,1> | <-1, 4, 4, -1> | <4, -2, 2, 0>>;
```

$$A0 := \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} \quad (1)$$

```
> Q0, R0 := QRDecomposition (A0, fullspan=true);
```

$$Q0, R0 := \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}, \begin{bmatrix} 2 & 3 & 2 \\ 0 & 5 & -2 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

```
> Lambda := DiagonalMatrix ([-74, 38, 2, 42]);
```

$$\Lambda := \begin{bmatrix} -74 & 0 & 0 & 0 \\ 0 & 38 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 42 \end{bmatrix} \tag{3}$$

```
> A[0] := Q0 . Lambda . Transpose(Q0);
```

$$A_0 := \begin{bmatrix} 2 & -18 & -38 & -20 \\ -18 & 2 & -20 & -38 \\ -38 & -20 & 2 & -18 \\ -20 & -38 & -18 & 2 \end{bmatrix} \tag{4}$$

```
> Eigenvalues (A[0]);
```

$$\begin{bmatrix} 2 \\ -74 \\ 38 \\ 42 \end{bmatrix} \tag{5}$$

Le jeu consiste maintenant à retrouver les valeurs propres à partir de A[0]

```
> A[0] := evalf(A[0]):
  N := 20:
  for k from 1 to N do
      Q[k], R[k] := QRDecomposition (A[k-1]);
      A[k] := R[k] . Q[k]
  end do:
  A[N];
```

$$[[-73.9999999829097, 0.000182077197506553, -0.00139575567553234, -9.39971736542978\,10^{-15}], [0.000182077197544666, 37.9999999997040, 2.26906681723089\,10^{-9}, 2.03552793231751\,10^{-15}], [-0.00139575567557495, 2.26907445346183\,10^{-9}, 41.9999999832057, \tag{6}$$

$-5.27745434669947\,10^{-15}$],
[$3.45983593922711\,10^{-30}$, $-9.57702192605121\,10^{-25}$,
$-1.43771927925894\,10^{-25}$, 2.00000000000000]]

La suite de matrices (A[k]) converge (sous certaines hypothèses) vers la matrice T de la factorisation de Schur
Comme A[0] est symétrique, la suite doit converger vers une matrice diagonale.

L'énoncé est très simple mais c'est un algorithme compliqué à prouver.

- Dans une véritable implantation, on n'a pas besoin de mémoriser toute la suite de matrices.
- On n'a pas non plus besoin d'expliciter les matrices Q[k].
- Les vecteurs « v » de la factorisation QR suffisent pour faire les multiplications par Q[k] sans expliciter Q[k]
- Il y a des optimisations qui permettent de mettre les matrices A[k] sous forme tridiagonale. Dans ce cas, la factorisation QR se simplifie (mise sous forme de Hessenberg).
- Il y a d'autres optimisations qui permettent de faire diminuer la dimension des matrices A[k] dès qu'une valeur propre est identifiée (mécanisme de déflation).

Je vais donner quelques éléments de preuve qui sont dans le poly.

Prop. 22 du poly.

Notons P[k] = Q[1] . Q[2] . ... . Q[k]
Alors on peut montrer que A[0] = P[k] . A[k] . P[k]^T

Ça montre que A[0] et A[k] sont semblables (puisque P[k] est une matrice orthogonale)

Ça montre aussi que si P[k] converge vers la matrice X (de la diagonalisation A[0] = X . Lambda . X^T avec X orthogonale) alors A[k] converge vers Lambda = T.

Effectivement : on a A[0] = X . Lambda . X^T = P[k] . A[k] . P[k]^T (prop 22).
Donc X^T . A[0] . X = (X^T . X) . Lambda . (X^T . X) = Lambda
Et si on suppose que P[k] = X alors on a X^T . A[0] . X = (X^T . P[k]) . A[k] . (P[k]^T . X) = A[k]
Donc Lambda = A[k].

Ce qui resterait à montrer, c'est que P[k] converge vers X.
L'idée, c'est que derrière l'algorithme QR, se cache la méthode de la puissance et la méthode de la puissance inverse.

Je vous propose de faire une pause. Je reviens sur Slack

La méthode de Hessenberg, ce sera pour un DM ;-)