Rapport de laboratoire 1: méthode de Gram-Schmidt

MTH8211

Ulrich Baron-Fournier

```
using Pkg
Pkg.activate("rapport_env")
Pkg.add("Test")
Pkg.add("LinearAlgebra")

using Test
using LinearAlgebra
```

Implémentation de Gram-Schmidt classique

```
using LinearAlgebra

function gram_schmidt(A::AbstractMatrix)
    n, m = size(A)
    Q = zeros(n, m)
    for j in 1:m
        v = A[:,j]
        for i in 1:j-1
            v -= (Q[:,i]' * A[:,j]) * Q[:,i]
        end
        norm_v = norm(v)
        if norm_v == 0
            error("Dépendance linéaire détectée : le vecteur $j est nul après projection.")
        end
        Q[:,j] = v / norm_v
    end
    return Q
end
```

```
gram_schmidt (generic function with 1 method)
```

Illustration de l'échec en cas de dépendance linéaire

```
try
    A_dep = [1 2; 2 4]
    gram_schmidt(A_dep)
catch e
    println("Erreur obtenue :")
    println(e)
end
```

```
Erreur obtenue :
ErrorException("Dépendance linéaire détectée : le vecteur 2 est nul après projection.")
```

Ici, la deuxième colonne est un multiple de la première, donc la famille est linéairement dépendante. L'algorithme échoue car il obtient un vecteur nul lors de l'orthogonalisation.

Implémentation de Gram-Schmidt plus robuste

Pour traiter le cas général, nous testons si la norme du vecteur à ajouter est inférieure à un seuil (tolérance atol). Les colonnes dépendantes (ou quasi-dépendantes) sont simplement ignorées :

```
using LinearAlgebra

function gram_schmidt_robuste(A::AbstractMatrix; atol=1e-12)
    n, m = size(A)
    Q = zeros(n, m)
    nb_vects = 0
    for j in 1:m
        v = A[:,j]
        for i in 1:nb_vects
            v -= (Q[:,i]' * A[:,j]) * Q[:,i]
        end
        norm_v = norm(v)
        if norm_v < atol
            @warn "Colonne $j ignorée (dépendance linéaire ou vecteur trop petit, norm = $norm
            continue
        end
        nb_vects += 1
        Q[:,nb_vects] = v / norm_v
    end
    return Q[:, 1:nb_vects]
end
```

```
gram_schmidt_robuste (generic function with 1 method)
```

Les avantages de cette méthode est que la fonction ne renvoie pas un message d'erreur si les colonnes sont dépendantes et cette fonction permet de traiter des matrices presques dépendantes (dans les cas où on retrouve des problèmes numériques). Les limites de cette modifications de la fonction sont le choix du paramètre atol qui peut influencer le résultat final. De plus, on ne peut pas savoir à l'avance combien de vecteurs seront retenus.

Démonstration de la nouvelle implémentation

```
# test.jl
using Test, LinearAlgebra
include("gram-schmidt.jl")
include("gram-schmidt-robuste.jl")

# Cas simple sans dépendance
A = [1 1; 0 1]
Q = gram_schmidt(A)
@test abs(norm(Q[:,1]) - 1) < 1e-12 # Norme colonne 1 =1
@test abs(norm(Q[:,2]) - 1) < 1e-12 # Norme colonne 2 = 1
@test abs(Q[:,1]' * Q[:,2]) < 1e-12 # Colonnes orthogonales

# Cas avec dépendance linéaire
A_dep = [1 2; 2 4]
@test_throws ErrorException gram_schmidt(A_dep) # Erreur pour gramschmidt non robuste

Q2 = gram_schmidt_robuste(A_dep)
@test size(Q2,2) == 1 # Une seule colonne survivante

# Cas tolérance
A_approx = [1 2+1e-13; 2 4+2e-13]
Q3 = gram_schmidt_robuste(A_approx, atol=1e-10)
@test size(Q3,2) == 1

println("Tous les tests sont passés.")
```

```
  Warning: Colonne 2 ignorée (dépendance linéaire ou vecteur trop petit, norm = 0.0)
  @ Main.Notebook C:\Users\ulric\gram-schmidt-robuste\gram-schmidt-robuste.jl:14
  Warning: Colonne 2 ignorée (dépendance linéaire ou vecteur trop petit, norm = 0.0)
  @ Main.Notebook C:\Users\ulric\gram-schmidt-robuste\gram-schmidt-robuste.jl:14
Tous les tests sont passés.
```