Práctica 3 – Matemática Aplicada. Grupo 1^oB

Método de Gram-Schmidt y mínimos cuadrados.

Índice

Producto escalar y base ortogonal.

Como vimos, para calcular el producto matricial o el producto escalar de dos vectores, se puede usar el comando `dot`.

Por ejemplo, si queremos calcular el producto escalar de los vectores $v_1 = (1, 2, -1)$ y $v_2 = (-2, 1, -1)$, podemos hacerlo de la siguiente manera.

Ejemplo 1.

```
from numpy import *

v1=array([1,2,-1])
v2=array([-2,1,-1])

print( dot(v1,v2) )
```

Ejercicio 2.

1. *Calcule el producto escalar de los vectores* $(2, 3, -1)$ *y* $(1, -2, 3)$.
2. *¿Son ortogonales los vectores* $(2, 2, 1)$ *y* $(2, -2, 1)$*? ¿y los vectores* $(2, 2, 1)$ *y* $(-1, 2, -2)$*?*
3. *Calcule el módulo del vector* $(1, 2, 3)$.

(Sol. -5. No ortogonales. Ortogonales. Aproximadamente 3.742).

Para comprobar si una base $B_U = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\}$ de un subespacio vectorial U está formada por vectores ortogonales basta calcular los productos escalares $\langle \mathbf{u}_i, \mathbf{u}_j \rangle$ para $i \neq j$.

Una forma sería formar una matriz con los vectores de la base como filas, y multiplicar esta matriz por su traspuesta.

$$\begin{pmatrix} \cdots & \mathbf{u}_1 & \cdots \\ \cdots & \mathbf{u}_2 & \cdots \\ & \cdots & \\ \cdots & \mathbf{u}_r & \cdots \end{pmatrix} \cdot \begin{pmatrix} \vdots & \vdots & \vdots \\ \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_r \\ \vdots & \vdots & \vdots \end{pmatrix}$$

Para esto, podemos definir la matriz como un array de Python `U`, de forma que los vectores del array serían los vectores `U[0]`, `U[1]`, ... Lo vemos a continuación.

Ejemplo 3.

```
from numpy import *
U=array([ [1,1,0],[1,2,0],[0,1,2]])

print( dot( U,transpose(U) ) )
```

Otra forma podría ser definir una matriz elemento a elemento de forma que en la posición (i, j) contuviera el valor $\langle \mathbf{u}_i, \mathbf{u}_j \rangle$.

Ejemplo 4.

```
from numpy import *
U=array([ [1,1,0],[1,2,0],[0,1,2] ])

[num_vectores,dimension]=shape(U)
matriz_gram=zeros([num_vectores,num_vectores])

for i in range(num_vectores):
    for j in range(num_vectores):
        matriz_gram[i,j]=dot(U[i],U[j])

print( matriz_gram )
```

En este ejemplo, la función `shape` permite determinar las dimensiones de la matriz `U` que contiene los vectores de la base B_U.

Ejercicio 5. *Compruebe si la base* $B_U = \{(1,1,1),(0,1,-1),(-1,\frac{1}{2},\frac{1}{2})\}$ *es ortogonal.*

(Sol. Sí, es ortogonal)

Método de ortogonalización de Gram-Schmidt.

En esta sección vamos a implementar el método de Gram-Schmidt en Python, donde se considera que los vectores se numeran comenzando en cero.

Así si tenemos un conjunto de vectores $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}\}$, el método de Gram-Schmidt requiere calcular los vectores

$$\mathbf{e}_j = \mathbf{u}_j - \frac{\langle e_0, u_j\rangle}{\|e_0\|^2}\mathbf{e}_0 - \frac{\langle e_1, u_j\rangle}{\|e_1\|^2}\mathbf{e}_1 - \ldots - \frac{\langle e_{j-1}, u_j\rangle}{\|e_{j-1}\|^2}\mathbf{e}_{j-1}$$

para $j = 0, \ldots, n-1$.

Para escribir este expresión en Python, podemos expresarla de la siguiente manera

$$\mathbf{e}_j = \mathbf{u}_j - \left(\frac{\langle e_0, u_j\rangle}{\|e_0\|^2}\mathbf{e}_0 + \frac{\langle e_1, u_j\rangle}{\|e_1\|^2}\mathbf{e}_1 + \ldots + \frac{\langle e_{j-1}, u_j\rangle}{\|e_{j-1}\|^2}\mathbf{e}_{j-1}\right).$$

De esta manera, la expresión que está dentro del paréntesis se puede implementar mediante un acumulador de suma. A continuación se da un código en Python que implementa este acumulador, primero se inicializa un vector `v` con ceros, y luego usando la variable `i` permite acumular los valores de

$$\frac{\langle \texttt{E[i]}, \texttt{U[j]}\rangle}{\langle \texttt{E[i]}, \texttt{E[i]}\rangle} * \texttt{E[i]}$$

Ejemplo 6.

```
v=zeros(dimension)
#Suma para i desde 0 hasta j-1
for i in range(j):
    v=v+dot(E[i],U[j])/dot(E[i],E[i])*E[i]
```

Este sumatorio forma parte del código que se propone para implementar el método de Gram-Schmidt.

Ejemplo 7 (`www.ugr.es/local/anpalom`).
`GramSchmidt.py`

Proyección ortogonal con la matriz de Gram.

En esta sección calculamos la proyección ortogonal de un vector $\mathbf{v} \in V$ en un subespacio U, es decir, calculamos $p_U(\mathbf{v})$. Para ello primero deduciremos un sistema ecuaciones para calcular la proyección, y luego implementaremos esas ecuaciones en Python.

Consideramos una base $B_U = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r\}$ de U, de forma que $p_U(\mathbf{v}) = a_1 \cdot \mathbf{u}_1 + a_2 \cdot \mathbf{u}_2 + \ldots + a_r \cdot \mathbf{u}_r$.

Para calcular los valores a_i para $i = 1, \ldots, r$, usamos que $\mathbf{v}$ se puede escribir de forma única como suma de un vector de U y otro de $U^\perp$. Como consecuencia, $\mathbf{v} - p_U(\mathbf{v})$ es ortogonal a todos los vectores de B_U.

En concreto, planteamos las ecuaciones

$$\langle \mathbf{v} - p_U(\mathbf{v}), \mathbf{u}_i \rangle = 0, \text{ para } i = 0, \ldots, r,$$

es decir

$$\langle p_U(\mathbf{v}), \mathbf{u}_i \rangle = \langle \mathbf{v}, \mathbf{u}_i \rangle \text{ para } i = 0, \ldots, r.$$

Como $p_U(\mathbf{v}) = a_1 \cdot \mathbf{u}_1 + a_2 \cdot \mathbf{u}_2 + \ldots + a_r \cdot \mathbf{u}_r$, las ecuaciones pueden plantearse como

$$\langle a_1 \cdot \mathbf{u}_1 + a_2 \cdot \mathbf{u}_2 + \ldots + a_r \cdot \mathbf{u}_r, \mathbf{u}_i \rangle = \langle \mathbf{v}, \mathbf{u}_i \rangle \text{ para } i = 0, \ldots, r,$$

y por linealidad

$$a_1 \cdot \langle \mathbf{u}_1, \mathbf{u}_i \rangle + a_2 \cdot \langle \mathbf{u}_2, \mathbf{u}_i \rangle + \ldots + a_r \cdot \langle \mathbf{u}_r, \mathbf{u}_i \rangle = \langle \mathbf{v}, \mathbf{u}_i \rangle \text{ para } i = 0, \ldots, r.$$

Así el sistema que tenemos que implementar tiene la forma

$$\begin{pmatrix} \langle \mathbf{u}_1, \mathbf{u}_1 \rangle & \langle \mathbf{u}_2, \mathbf{u}_1 \rangle & \ldots & \langle \mathbf{u}_r, \mathbf{u}_1 \rangle \\ \vdots & \vdots & & \vdots \\ \langle \mathbf{u}_1, \mathbf{u}_i \rangle & \langle \mathbf{u}_2, \mathbf{u}_i \rangle & \ldots & \langle \mathbf{u}_r, \mathbf{u}_i \rangle \\ \vdots & \vdots & & \vdots \\ \langle \mathbf{u}_1, \mathbf{u}_r \rangle & \langle \mathbf{u}_2, \mathbf{u}_r \rangle & \ldots & \langle \mathbf{u}_r, \mathbf{u}_r \rangle \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_r \end{pmatrix} = \begin{pmatrix} \langle \mathbf{v}, \mathbf{u}_1 \rangle \\ \vdots \\ \langle \mathbf{v}, \mathbf{u}_i \rangle \\ \vdots \\ \langle \mathbf{v}, \mathbf{u}_r \rangle \end{pmatrix}.$$

Para implementar este sistema en Python, primero importámos el módulo `numpy` e inicializamos variables.

Ejemplo 8 (Parte 1 de 4).

```
from numpy import *

BU=array([ [1,2,0], [0,3,1] ])
v=array([3,-2,4])

[num_vectores,dimension]=shape(BU)
matriz_gram=zeros([num_vectores,num_vectores])
b=zeros(num_vectores)
```

Así `BU` es la base B_U, con lo que los vectores de la base sern `BU[0]`, `BU[1]`, ... En `v` se define el vector que se va a proyectar. En la variable `num_vectores` guardamos el nùmero de vectores de la base, y en `dimension` el número de componentes de los vectores de la base. Con estos valores definimos `matriz_gram` donde almacenaremos los valores de $\langle \mathbf{u}_i, \mathbf{u}_j \rangle$, es decir `dot(BU[i],BU[j])`. En el vector `b` almacenaremos los términos independientes del sistema lineal, esto es, los valores de $\langle \mathbf{v}, \mathbf{u}_i \rangle$, en Python `dot(v,BU[i])`.

En los siguiente bucles definimos el vector independiente `b` y la matriz del sistema `matriz_gram`.

Ejemplo 9 (Parte 2 de 4).

```
for i in range(num_vectores):
    b[i]=dot(v,BU[i])
    for j in range(num_vectores):
        matriz_gram[i,j]=dot(BU[i],BU[j])
```

Ahora hallamos la solución del sistema, y almacenamos en `coeficientes_a` los valores de a_i. Con estos valores hallamos
$p_U(v) = \sum a_i \cdot \mathbf{v}_i =$
`coeficientes_a[0]*BU[0] + coeficientes_a[1]*BU[1] + ....`

Para esto inicializamos a cero un acumulador de suma, y con un bucle vamos sumando todos los `coeficientes_a[i]*BU[i]`.

Ejemplo 10 (Parte 3 de 4).

```
coeficientes_a=( linalg.solve(matriz_gram, b) )

pU=zeros(dimension)
for i in range(num_vectores):
    pU=pU+coeficientes_a[i]*BU[i]
```

Por último mostramos los resultados, primero los coeficientes a_i, y luego el vector proyectado $p_U(\mathbf{v})$.

Ejemplo 11 (Parte 4 de 4).

```
print('Coeficientes:')
print(coeficientes_a)

print('Proyeccion pU(v):')
print(pU)
```

Con las cuatro partes que hemos mostrado, tenemos un programa para calcular la proyección ortogonal de un vector sobre un subespacio vectorial del que conocemos una base.

Ejemplo 12 (`www.ugr.es/local/anpalom`).
`ProyeccionGram.py`

Ejercicio 13. *Halle la proyección ortogonal del vector* $\mathbf{v} = (0, 3, 0)$ *en el plano generado por los vectores* $\mathbf{u}_1 = (1, 1, 1)$ *y* $\mathbf{u}_2 = (1, 0, 1)$.

(Sol. $p_U(\mathbf{v}) = (0, 3, 0)$*, aproximadamente* $(4{,}4408921 \cdot 10^{-16}, 3{,}0000000, 4{,}4408921 \cdot 10^{-16})$*.)*

Ejercicio 14. *Sea* $v = (1, 1, 1)$ *y sea* U *el espacio vectorial generado por los vectores* $(-1, 0, 2)$ *y* $(-1, 3, 0)$.

1. *¿Es* $pU(v) = (-\frac{17}{49}, \frac{27}{49}, \frac{16}{49})$*?.*
2. *Halle* $v - pU(v)$.

3. Usando el apartado anterior, halle la distancia entre v y U.

(Sol. Si. Aproximadamente $(1{,}34693878, 0{,}44897959, 0{,}67346939)$. *Aproximadamente 1.5714285714285714.)*

Ejercicio 15. *Dentro del bucle de la parte 2, ¿puede calcularse* `matriz_gram[i,j]` *como* `dot(BU[j],BU[i])`*?*

Ejercicio de proyección ortogonal de un polinomio.

En esta sección daremos los pasos necesarios para hallar la proyección ortogonal de un polinomio en un subespacio vectorial. Nos planteamos el siguiente ejercicio.

En el espacio vectorial de los polinomios de grado menor o igual que 3 definidos en el intervalo $[0,1]$ ($\mathcal{P}_3(x,[0,1])$), considera el subespacio vectorial U dado por

$$U = \{a_0 + a_1x + a_2x^2 + a_3x^3 \in \mathcal{P}_3(x) : a_0 - 2a_1 = 0, a_0 + a_1 + a_3 = 0\}.$$

1. Halla una base de U.
2. Calcula una base ortogonal de U.
3. Calcula la proyección ortogonal del polinomio $-1 + 2x + 3x^2 + x^3$ sobre U.

Para calcular una base de U, en primer lugar nos planteamos el sistema lineal formado por las ecuaciones que definen U.

$$\begin{cases} a_0 - 2a_1 = 0, \\ a_0 + a_1 + a_3 = 0. \end{cases}$$

En el código de debajo, usamos `solve_linear_system` para resolver el sistema lineal, y obtenemos valores de a_1 y a_0 en función del parametro a_3. En concreto se obtiene

$$a_1 = -\frac{1}{3}a_3, \quad \text{y} \quad a_0 = -\frac{2}{3}a_3.$$

Al sustituir estos valores en un polinomio $p(x)$ de $\mathcal{P}_3(x)$, tenemos que

$$p(x) \in U \Leftrightarrow p(x) = \left(-\frac{2}{3}a_3\right) + \left(-\frac{1}{3}a_3\right)x + a_2x^2 + a_3x^3.$$

En Python esta substitución se hace mediante `subs`.

Para hallar una base de U, usamos `collect` con lo que sacamos factor común a_3 y a_2,

$$p(x) = a_3\left(-\frac{2}{3} - \frac{1}{3}x + x^3\right) + a_2x^2 = a_3 \cdot p_1(x) + a_2 \cdot p_2(x).$$

Así pues una base de U es $B = \{-\frac{2}{3} - \frac{1}{3}x + x^3, x^2\}$.

Ejemplo 16 (Parte 1 de 3).

```
from sympy import symbols, Matrix, solve_linear_system, \
    integrate, solve, N
x=symbols('x')
a0, a1, a2, a3 = symbols('a0 a1 a2 a3')
A = Matrix( [[1, -2, 0, 0, 0], [0, 3, 0, 1, 0]] )

print('Solucion del sistema:')
print( solve_linear_system(A, a0, a1, a2, a3) )

polinomio=(a0+a1*x+a2*x**2+a3*x**3).subs(a1,-a3/3) \
    .subs(a0,-2*a3/3)

print('Un polinomio de U generico')
print(  polinomio.collect(a3)  )

p1=(3*x**3 - x - 2)/3
p2=x**2
```

Una vez obtenida esa base, nos planteamos la segunda parte del ejercicio que es calcular una base ortogonal de U. Lo haremos usando el método de Gram-Schmidt, para lo cual definimos dos polinomios $e_1(x)$ y $e_2(x)$ de la siguiente forma

$$\begin{aligned} e_1 &= p_1, \\ e_2 &= p_2 - \alpha \cdot p_1 \quad \text{con } \alpha \text{ tal que } \langle p_2 - \alpha \cdot p_1, p_1\rangle = 0. \end{aligned}$$

Es decir, buscamos el valor de α que hagan que e_1 y e_2 sean ortogonales. En Python, definimos un nuevo símbolo `alpha`, definimos los vectores `e1` y

`e2`, y planteamos el producto escalar de los dos para calcular

$$\texttt{productoescalar} = \int_0^1 \texttt{e1}(x)*\texttt{e2}(x)\,dx$$

donde la integral se realiza con `integrate`.

Usamos la función `solve` para despejar el valor de `alpha` en la ecuación `productoescalar`$= 0$. Como resultado obtenemos el valor $\alpha =$`-525/1436`, y con este definimos de nuevo el polinomio e_2.

Ejemplo 17 (Parte 2 de 3).

```
alpha=symbols('alpha')

e1=p1
e2=p2-alpha*p1

print('Producto escalar e1 por e2')
productoescalar= integrate( e1*e2,(x,0,1) )
print( productoescalar )

solucion=solve( productoescalar , alpha)
print( solucion )

e2=p2-(-525*p1/1436)
```

Por último calculamos la proyección ortogonal del polinomio $v = -1 + 2x + 3x^2 + x^3$ sobre U. Para ello usamos la fórmula

$$p_U(v) = \frac{\langle v, e_1\rangle}{\|e_1\|^2} e_1 + \frac{\langle v, e_1\rangle}{\|e_1\|^2} e_2.$$

Así pues para resolver el último apartado del ejercicio, definimos el vector v y calculamos $\langle v, e_1\rangle$, $\|e_1\|^2$, $\langle v, e_2\rangle$ y $\|e_2\|^2$. Vemos cómo se hace en Python en el siguiente código, donde mostramos el valor exacto de `pu` y una aproximación obtenida con `N`.

Ejemplo 18 (Parte 3 de 3).

```
v=-1+2*x+3*x**2+x**3
```

```
numerador1=integrate(v*e1,(x,0,1))
denominador1=integrate(e1*e1,(x,0,1))
numerador2=integrate(v*e2,(x,0,1))
denominador2=integrate(e2*e2,(x,0,1))

pu=numerador1*e1/denominador1 + numerador2*e2/denominador2

print(pu)
print( N(pu) )
```

Con las tres partes de código se dan todos los pasos para resolver el ejercicio planteado.

Ejemplo 19 (`www.ugr.es/local/anpalom`). `ProyeccionPolinomio.py`

Ejercicio 20. *Considere los vectores* $\mathbf{u}_1 = (1,1,2)$ *y* $\mathbf{u}_2 = (-1,-1,3)$. *Utilice el método de Gram-Schmidt para hallar dos vectores* $\mathbf{e}_1$ *y* $\mathbf{e}_2$ *de forma que* $\mathbf{e}_1 = \mathbf{u}_1$, $\mathbf{e}_2 = \mathbf{u} - \alpha \cdot \mathbf{e}_1$ *y los vectores* $\mathbf{e}_1$ *y* $\mathbf{e}_2$ *sean ortogonales. Compruebe que los dos vectores obtenidos son ortogonales.*

Indicación: Puede comenzar el programa de la siguiente manera.

```
from numpy import array, dot
from sympy import symbols, solve

u1=array([1,1,2])
u2=array([-1,-1,3])

alpha=symbols('alpha')
```

(Sol. $\mathbf{e}_2 = (-\frac{5}{3}, -\frac{5}{3}, \frac{5}{3})$, *aproximadamente* `[-1.66666667, -1.66666667, 1.66666667]`*)*

Ejercicio 21. *Considere los vectores* $\mathbf{e}_1 = (1,-1,0)$ *y* $\mathbf{e}_2 = (\frac{1}{2}, \frac{1}{2}, -1)$.

1. *Compruebe que son ortogonales.*
2. *Halle la proyección ortogonal del vector* $v = (1,1,1)$ *sobre el plano generado por los vectores* $\mathbf{e}_1$ *y* $\mathbf{e}_2$.

3. *Halle la proyección ortogonal del vector* $v = (1, 2, 3)$ *sobre el plano generado por los vectores* $\mathbf{e}_1$ *y* $\mathbf{e}_2$*.*

(Sol. $p(1, 1, 1) = (0, 0, 0)$, $p(1, 2, 3) = (-1, 0, 1)$.)

Ejercicio 22. *Considere* $\mathcal{P}_2(x, [0, 1])$*, el espacio de funciones de los polinomios de grado 2 con el producto escalar* $\langle p, q\rangle = \int_0^1 p(x) \cdot q(x)\, dx$*. Considere los polinomios* $p(x) = 6x^2 - 6x + 1$*,* $q(x) = 2x - 1$*.*

Halle $\langle p(x), q(x)\rangle$*.*

(Sol. 0)