determ-excep

May 5, 2021

Definiciones originales determinante recursivo sin excepciones

[1]:
```
def matriz_adjunta(m,i,j):
    """
    devuelve la matriz que se obtiene de m eliminando la fila i y la columna j
    0<=i< len(m), 0<=j<len(m[0])
    """
    n_rows = len(m)
    n_cols = len(m[0])
    result = []
    for row in range(n_rows):
        if row!=i:
            new_row = []
            for col in range(n_cols):
                if col != j:
                    new_row.append(m[row][col])
            result.append(new_row)
    return result

def det(m):
    """==
    m es una matriz cuadrada
    """
    n = len(m)
    if n == 1:
        return m[0][0]
    else:
        result = 0
        sgn = 1
        for j in range(len(m)):
            result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
            sgn = - sgn
        return result
```

Aunque no tenga efectos, aquí elimino la "restricción" de que `m` sea cuadrada

[2]:
```
def det(m):
    n = len(m)
    if n == 1:
```

```
        return float(m[0][0])
    else:
        result = 0
        sgn = 1
        for j in range(len(m)):
            result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
            sgn = - sgn
        return result
```

[3]:
```
det([[1,0,0],[0,1,0],[0,0,5]])
```

[3]:
```
5.0
```

Hago una llamada errónea que produce TypeError.
Capturo cualquier error e imprimo el mensaje.
Le ejecución prosigue, pero `res` "vale" `None`

[4]:
```
try:
    res=det([[1,0,0],[0,'a',0],[0,0,5]])
except:
    print('datos erróneos en matriz')
```

```
datos erróneos en matriz
```

[5]:
```
print(res)
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-5-a90dea80f461> in <module>
----> 1 print(res)


NameError: name 'res' is not defined
```

Incorporo el manejador de excepciones al código de `det`

[6]:
```
def det(m):
    try:
        n = len(m)
        if n == 1:
            return float(m[0][0])
        else:
            result = 0
            sgn = 1
```

```
            for j in range(len(m)):
                result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
                sgn = - sgn
            return result
    except:
        print('datos erróneos en matriz')
        return None
```

Observad que el mensaje sale varias veces por culpa de que `det` es recursivo.
Al final `res` sigue siendo `None`

[7]:
```
res=det([[1,0,0],[0,0,5],[0,'a',0]])
print(res)
```

```
datos erróneos en matriz
datos erróneos en matriz
datos erróneos en matriz
None
```

[8]:
```
det([[2,2,0,0],[0,2,'b',0],[0,0,5,0],[0,0,0,9]])
```

```
datos erróneos en matriz
datos erróneos en matriz
```

[9]:
```
det([[2,1],[0,'b']])
```

```
datos erróneos en matriz
datos erróneos en matriz
```

[10]:
```
det([['b']])
```

```
datos erróneos en matriz
```

Defino una excepciónnueva `Mat_no_cuadrada`

[11]:
```
class Mat_no_cuadrada(Exception):
    pass
```

Incorporo su manejo en `det`

[12]:
```
def det(m):
    try:
        n = len(m)
        if not (n==len(m[0])):
            raise Mat_no_cuadrada
        if n == 1:
            return float(m[0][0])
        else:
            result = 0
```

```
            sgn = 1
            for j in range(len(m)):
                result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
                sgn = - sgn
            return result
    except Mat_no_cuadrada:
        print('matriz no cuadrada')
    except:
        return None
```

[13]:
```
det([[1,0,0],[0,1,0],[0,0,5]])
```

[13]:
```
5.0
```

[14]:
```
res=det([[1,0],[0,1,0],[0,0,5]])
print(res)
```

```
matriz no cuadrada
None
```

El caso siguiente no estaba contemplado.
Sin embargo, se producen errores que he capturado con el segundo "manejador".

[15]:
```
res=det([[1,0,1],[0,1],[0,0,5]])
print(res)
```

```
None
```

Corrijo el código mirando que todas las filas tengan tantos elementos como filas tenga `m`.

[16]:
```
def det(m):
    try:
        n = len(m)
        if not (n==len(m[0])):
            raise Mat_no_cuadrada
        if n == 1:
            return float(m[0][0])
        else:
            result = 0
            sgn = 1
            for j in range(len(m)):
                result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
                sgn = - sgn
            return result
    except Mat_no_cuadrada:
        print('matriz no cuadrada')
    except:
        return None
```

Añado el parámetro `j` que me indica una fila (podría haber más detrás) errónea de `m`.
Pero no capturo el error `Mat_no_cuadrada` dejando que "salga fuera".

[17]:
```
def det(m):
    try:
        n = len(m)
        for j in range(len(m)):
             if not (n==len(m[j])):
                raise Mat_no_cuadrada(j)
        if n == 1:
            return float(m[0][0])
        else:
            result = 0
            sgn = 1
            for j in range(len(m)):
                result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
                sgn = - sgn
            return result
    except TypeError:
        return None
```

[18]:
```
try:
    det([[1,0],[0,1,0],[0,0,5]])
except Mat_no_cuadrada:
    print('matriz no cuadrada')
```

```
matriz no cuadrada
```

[19]:
```
det([[1,0,0],[0,1],[0,0,5]])
```

```
␣
↪---------------------------------------------------------------------------

Mat_no_cuadrada                           Traceback (most recent call␣
↪last)

<ipython-input-19-4087fff0b0d4> in <module>
----> 1 det([[1,0,0],[0,1],[0,0,5]])


<ipython-input-17-ff147250c509> in det(m)
      4         for j in range(len(m)):
      5              if not (n==len(m[j])):
----> 6                 raise Mat_no_cuadrada(j)
      7         if n == 1:
      8             return float(m[0][0])


Mat_no_cuadrada: 1
```

Capturo otra vez `Mat_no_cuadrada` en `det`usando el parámetro `j`

[20]:
```
def det(m):
    try:
        n = len(m)
        for j in range(len(m)):
             if not (n==len(m[j])):
                raise Mat_no_cuadrada(j)
        if n == 1:
            return float(m[0][0])
        else:
            result = 0
            sgn = 1
            for j in range(len(m)):
                result = result + sgn * m[0][j] * det(matriz_adjunta(m,0,j))
                sgn = - sgn
            return result
    except Mat_no_cuadrada as j:
        print('matriz no cuadrada por culpa de la fila:',j)
    except:
        return None
```

[21]:
```
det([[1,0,0],[0,1],[0,0,5]])
```

```
matriz no cuadrada por culpa de la fila: 1
```

Aquí tenéis la implementación recursiva de `det` usando dirtectamente "trozos" de `m`, sin copiar los adjuntos.

[22]:
```
def matriz_adjunta(m,i,j):
    """
    devuelve la matriz que se obtiene de m eliminando la fila i y la columna j
    0<=i< len(m), 0<=j<len(m[0])
    """
    n_rows = len(m)
    n_cols = len(m[0])
    result = []
    for row in range(n_rows):
        if row!=i:
            new_row = []
            for col in range(n_cols):
                if col != j:
                    new_row.append(m[row][col])
            result.append(new_row)
    return result



def det_trozo(m,lf,lc):
    if len(lf)== 1:
```

```
        return m[lf[0]][lc[0]]
    else:
        result = 0
        sgn = 1
        nlf=lf[1:]
        for j in range(len(lc)):
            nlc=lc[:]
            nlc.pop(j)
            result = result + sgn * m[lf[0]][lc[j]] * det_trozo(m,nlf,nlc)
            sgn = - sgn
        return result

def det(m):
    """==
    m es una matriz cuadrada
    """
    n = len(m)
    lf=[]
    for j in range(len(m)):
        lf.append(j)
    return det_trozo(m,lf,lf)
```

[23]:
```
det([[2,2,0,0],[0,2,0,0],[0,0,5,0],[0,0,0,9]])
```

[23]:
```
180
```

[24]:
```
det([[0,5],[9,3]])
```

[24]:
```
-45
```

Os dejo aquí una copia para que añadáis vosotros la detección y tratamiento de errores a imagen y semejanza de lo hecho arriba con la implementación que copia los adjuntos.

[25]:
```
def matriz_adjunta(m,i,j):
    """
    devuelve la matriz que se obtiene de m eliminando la fila i y la columna j
    0<=i< len(m), 0<=j<len(m[0])
    """
    n_rows = len(m)
    n_cols = len(m[0])
    result = []
    for row in range(n_rows):
        if row!=i:
            new_row = []
            for col in range(n_cols):
                if col != j:
                    new_row.append(m[row][col])
            result.append(new_row)
    return result
```

```
def det_trozo(m,lf,lc):
    if len(lf)== 1:
        return m[lf[0]][lc[0]]
    else:
        result = 0
        sgn = 1
        nlf=lf[1:]
        for j in range(len(lc)):
            nlc=lc[:j]+lc[j+1:]
            result = result + sgn * m[lf[0]][lc[j]] * det_trozo(m,nlf,nlc)
            sgn = - sgn
        return result

def det(m):
    """==
    m es una matriz cuadrada
    """
    n = len(m)
    lf=[]
    for j in range(len(m)):
        lf.append(j)
    return det_trozo(m,lf,lf)
```

[ ]: