Ejemplo mínimo para usar NAcAL en un documento de LaTeX

Marcos Bujosa

12 de junio de 2024

1. Primeros pasos del asunto

1.1. Preámbulo

En el preámbulo del documento de LaTeX hay que cargar el paquete `pythontex` e indicar a `pythontex` que debe usar el módulo NAcAL... El preámbulo de este documento incluye lo siguiente:

```
\usepackage{pythontex}
\pythontexcustomc{pyconsole}{from nacal import *}
\pythontexcustomc{py}{from nacal import *}
```

1.2. Compilación

Para compilar el documento es necesario dar varias pasadas. Como mínimo debemos ejecutar estas tres órdenes

```
pdflatex ParaMisCompis.tex

pythontex --interpreter python:python3 ParaMisCompis.tex

pdflatex ParaMisCompis.tex
```

donde `ParaMisCompis.tex` es el nombre del fichero que queréis compilar. Si vuestro documento tiene enlaces internos, índice de contenidos, glosario, bibliografía, etc. hay que compilar más veces y ejecutar más comandos para que todo esté OK. Como éste es un ejemplo mínimo, con esos tres comandos es sufi.

2. Uso de Python

Lo mejor es mirar la documentación de `pythontex` (`https://www.ctan.org/pkg/pythontex`)

Otra cosa importante es que hay que tener en cuenta que en Python los espacios (o tabuladores) delante de los comandos son importantes. Por tanto el código Python debe empezar al comienzo de la línea. Por ejemplo: este código es OK

```
C = Matrix([[1,1,2,4],[1,0,3,4]])
```

pero este otro daría error

```
  C = Matrix([[1,1,2,4],[1,0,3,4]])
```

(nótese que hay dos espacios en blanco al inicio).

El código se puede incluir en el texto con `\py`. Por ejemplo

```
Dos más dos son $\py{2+2}$.
```

genera el siguiente texto:

> Dos más dos son 4.

NAcAL define una representación LaTeX para la mayoría de objetos que implementa (sistemas, vectores, matrices, transformaciones elementales, subespacios afines, etc.) pero Python solo usa dicha representación con los Notebooks de Jupyter. Si no se indica explícitamente, Python no usa la representación LaTeX. Por tanto, con

```
Sea la matriz $\py{ repr(Matrix([[1,1,2,4],[1,0,3,4]])) }$.
```

obtenemos el texto

> Sea la matriz $Matrix([Vector([1,1]), Vector([1,0]), Vector([2,3]), Vector([4,4])])$.

Que es correcto pero muy feo.

Si queremos que las cosas se pinten de manera bonita hay que decir explícitamente a Python que use la representación LaTeX con el comando `latex()`. Por ejemplo, el código:

```
Sea la matriz $\py{ latex( Matrix([[1,1,2,4],[1,0,3,4]]) ) }$.
```

nos escribe

Sea la matriz $\begin{bmatrix} 1 & 1 & 2 & 4 \\ 1 & 0 & 3 & 4 \end{bmatrix}$.

Hay otra forma de trabajar (que es la que uso con más frecuencia). El entorno `pycode` ejecuta código de Python pero no muestra el resultado (hay que mirar la documentación de Pythontex). Así, con

```
\begin{pycode}
C = Matrix([[1,1,2,4],[1,0,3,4]])
\end{pycode}
```

ya hemos definido la matriz **C**. Ahora podemos escribir

```
Sea la matriz $\py{latex( C )}$, que tiene rango $\py{C.rango()}$ y
cuya forma escalonada por filas es $\py{ latex(C.U()) }$.
```

y obtener

Sea la matriz $\begin{bmatrix} 1 & 1 & 2 & 4 \\ 1 & 0 & 3 & 4 \end{bmatrix}$, que tiene rango 2 y cuya forma escalonada por filas es $\begin{bmatrix} 1 & 1 & 2 & 4 \\ 0 & -1 & 1 & 0 \end{bmatrix}$.

3. Uso de NAcAL

Algunos procedimientos del curso (como la eliminación) dan como resultado una matriz, pero también muestran como llegar al resultado. Por ejemplo, podemos escalonar por columnas la matriz **C**

```
\begin{displaymath}
  \py{latex( Elim(C) ) }
\end{displaymath}
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Pero solo nos ha mostrado el resultado final (es decir, la matriz escalonada por eliminación). Afortunadamente NAcAL guarda el código LaTeX de los pasos dados para llegar al resultado en el atributo `.tex`. Como dicho atributo ya es código LaTeX, no hay que usar el comando `latex()` dentro de Python.

```
\begin{displaymath}
  \py{Elim(C).tex}
\end{displaymath}
```

nos muestra lo siguiente

$$\begin{bmatrix} 1 & 1 & 2 & 4 \\ 1 & 0 & 3 & 4 \end{bmatrix} \xrightarrow[\substack{[(-2)\mathbf{1}+\mathbf{3}]\\ [(-4)\mathbf{1}+\mathbf{4}]}]{\boldsymbol{\tau}[(-1)\mathbf{1}+\mathbf{2}]} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 \end{bmatrix} \xrightarrow{\boldsymbol{\tau}[(1)\mathbf{2}+\mathbf{3}]} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Hay muchos objetos molestos para escribir en LaTeX pero que resultan inmediatos con NAcAL. Por ejemplo el conjunto de soluciones $\mathbf{C}\boldsymbol{x} = \mathbf{0}$

```
\begin{displaymath}
  \py{ latex( C.espacio_nulo() ) }
\end{displaymath}
```

nos muestra lo siguiente

$$\left\{ \boldsymbol{v} \in \mathbb{R}^4 \;\middle|\; \exists \boldsymbol{p} \in \mathbb{R}^2,\ \boldsymbol{v} = \begin{bmatrix} -3 & -4 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{p} \right\} = \left\{ \boldsymbol{v} \in \mathbb{R}^4 \;\middle|\; \begin{bmatrix} 0 & -3 & 3 & 0 \\ 1 & 3 & 0 & 4 \end{bmatrix} \boldsymbol{v} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$$

Si solo estamos interesados en escribir las ecuaciones paramétricas

```
\begin{displaymath}
  \py{ C.espacio_nulo().EcParametricas() }
\end{displaymath}
```

nos muestra lo siguiente

$$\left\{ \boldsymbol{v} \in \mathbb{R}^4 \;\middle|\; \exists \boldsymbol{p} \in \mathbb{R}^2,\ \boldsymbol{v} = \begin{bmatrix} -3 & -4 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{p} \right\}$$

(escribiendo esto me doy cuenta de que no he sido sistemático con los nombres... `C.espacio_nulo().EcParametricas()` ya es código LaTeX, y por el nombre no está claro... quizá debería darle una vuelta a eso).

En ocasiones he elegido otra representación para lo mismo. Por ejemplo, la clase `Homogenea` resuelve el sistema de ecuaciones homogéneo

```
\begin{displaymath}
  \py{ latex( Homogenea(C) )}
\end{displaymath}
```

nos muestra lo siguiente

$$\mathcal{L}\left(\left[\begin{pmatrix}-3\\1\\1\\0\end{pmatrix};\quad\begin{pmatrix}-4\\0\\0\\1\end{pmatrix};\right]\right)$$

Y si queremos ver cómo lo ha calculado...

```
\begin{displaymath}
  \py{Homogenea(C).tex }
\end{displaymath}
```

nos muestra lo siguiente

$$\left[\begin{array}{cccc}1&1&2&4\\1&0&3&4\\\hline1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{array}\right]\xrightarrow[\substack{[(-1)\mathbf{1}+\mathbf{2}]\\[(-2)\mathbf{1}+\mathbf{3}]\\[(-4)\mathbf{1}+\mathbf{4}]}]{\boldsymbol{\tau}}\left[\begin{array}{cccc}1&0&0&0\\1&-1&1&0\\\hline1&-1&-2&-4\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{array}\right]\xrightarrow[[(1)\mathbf{2}+\mathbf{3}]]{\boldsymbol{\tau}}\left[\begin{array}{cccc}1&0&0&0\\1&-1&0&0\\\hline1&-1&-3&-4\\0&1&1&0\\0&0&1&0\\0&0&0&1\end{array}\right]$$

Como podéis ver, escribir con Pythontex + NAcAL es bastante productivo, ahorra mucho trabajo y evita muchos errores. **!Espero que os animéis a usarlo!**... que yo sepa no hay nada igual

Como podéis imaginar la inversión de tiempo, trabajo y aprendizaje ha sido bastante considerable... sería un desperdicio que solo lo use yo.

Además, el código es abierto, así que podéis añadir lo que le falte.