Comenzando con PDL: un lenguaje para el cálculo numérico

Xavier Calbet*
e–mail: xca@ll.iac.es

Octubre de 2000

Esta ponencia forma parte del III Congreso Hispalinux[1] virtual, celebrado en Noviembre de 2000. Puedes participar en él en la sección de eventos de BarraPunto[2]

Resumen

Se presenta un breve tutorial de PDL, "Perl Data Language", para dar a conocer esta potente herramienta de cálculo numérico basada en Perl.

Asimismo se presenta el manual que ha escrito el autor para iniciarse en PDL: "Beginning PDL".

1 Introducción

PDL es un lenguaje de programación orientado al cálculo numérico. Sus siglas significan "Perl Data Language", y como su nombre indica está basado en el lenguaje de programación Perl. PDL es una extensión de Perl orientada al tratamiento numérico de datos. Puede hacer cálculos numéricos matriciales de una forma rápida.

Antes de continuar explicando lo que es PDL, daremos una breve idea de lo que es Perl.

1.1 Introducción a Perl

Perl es la abreviación de "Practical Extraction and Report Language". Es lo que se conoce como un lenguaje "script", es decir, uno en el que no hace falta compilar el programa escrito o "script". En la práctica se compila parcialmente al comienzo de su ejecución. Ha sido diseñado para "hacer las cosas fáciles fáciles y las cosas difíciles posibles".

Veamos algunas de las ventajas y desventajas de Perl:

- Es un buen lenguaje "pegamento". Se pueden juntar varios programas de una forma sencilla para alcanzar una meta determinada. Los usuarios de Windows agradecerán esta propiedad ya que normalmente adolecen de un buen lenguaje tipo "script".
- Es relativamente rápido para un lenguaje tipo "script".
- Está disponible en múltiples plataformas y sistemas operativos. De hecho funciona bajo diferentes sabores de UNIX, Linux y todo tipo de Windows. Un programa que se escriba teniendo en cuenta la compatibilidad puede ser escrito en una plataforma y ejecutado en otra.
- El desarrollo de aplicaciones es muy rápido.
- Hay una colección enorme de módulos que pueden ser incorporados a cualquier "script" de Perl. Están disponibles en el CPAN[3] ("Comprehensive Perl Archive Network"). En particular existe una extensión para cálculo numérico denominada PDL.

[1] `http://congreso.hispalinux.es`

[2] `http://barrapunto.com/eventos`

[3] `http://www.perl.com/CPAN`

- Perl es gratuito. Mucho más que eso, es “Software Libre”. Esto quiere decir que el código fuente está disponible para que cualquiera lo pueda ver o modificar, y lo que es más importante, siempre lo estará. Aunque nunca pretendas cambiar el código, es importante disponer de la posibilidad de hacerlo, ya que siempre se podrá contratar a una tercera persona para que lo modifique en el caso de que haya un error, y debería ser posible solucionarlo.
- Le otorga al programador mucha libertad para que haga el programa como quiera. Tal como dice el eslogan de Perl “Hay más de una forma de hacerlo”.

Algunas de las desventajas de Perl son:

- Es lento para algunas aplicaciones, como programación a bajo nivel, escribiendo un “driver” para una aplicación o corriendo modelos numéricos de cálculo intensivo. Si bien se pueden insertar subrutinas FORTRAN o C en Perl, teniendo lo mejor de los dos mundos, pero con algo más de complejidad.
- La libertad que se le otorga al programador puede significar que el resultado sea un programa ilegible. Si no se escribe con cuidado puede llegar a ser difícil de leer. De hecho hay un concurso de Perl ofuscado.
- No se pueden compilar programas Perl. Aunque actualmente se está desarrollando un compilador que realice esta tarea y hay uno comercial disponible para la plataforma Windows. Este tema no es tan crítico como suena, los programas Perl no correrán mucho más rápidos cuando se compilen, la única ventaja está en la desaparición de la fase inicial de compilación al correr la aplicación.
- Utiliza muchos recursos de la máquina. Esto significa que no es tan ligero como un programa en C, pero en la práctica es ligero comparado con la potencia de computación de los ordenadores actuales.

1.2 Introducción a PDL

Tal como hemos mencionado, PDL es un lenguaje de cálculo numérico basado en Perl con la posibilidad de manipular matrices n–dimensionales de forma rápida. Se puede conseguir gratuitamente en la red en la página principal de PDL[4]

Veamos ahora algunas ventajas y desventajas de PDL. Las principales ventajas de PDL son:

- Es gratuito y es “software libre”.
- Permite el cálculo con matrices y con mayores dimensiones, no se detiene en la segunda dimensión, sino que permite realizar cálculos con más dimensiones. En este sentido está muy bien diseñado y escala hacia arriba muy bien.
- Es muy rápido.
- La salida gráfica es muy buena. Las gráficas bidimensionales están basadas en PGPLOT, el cual es muy adecuado tanto para figuras con calidad de publicación como para figuras del día a día. Las gráficas tridimensionales, basadas en la librería MESA, aún no tienen una calidad de publicación, pero son muy buenas para representar parámetros en tres dimensiones de una forma sencilla y permite la rotación en tiempo real de los datos.

Las principales desventajas de PDL son:

- La librería numérica disponible para realizar cálculos específicos no es muy grande. Esta faceta está cambiando rápidamente a medida que más librerías numéricas se van incluyendo en PDL como por ejemplo GSL o SLATEC.
- No tiene soporte para cálculo de números complejos de forma nativa. En la versión 2.004 de PDL se ha incluido una paquete de cálculo complejo, pero sólo permite realizar aritmética compleja simple.
- Programar en PDL requiere un pensamiento abstracto mayor que en otros lenguajes para poder extraer toda su potencia. En algunos casos también ocurre que no hay una forma sencilla de utilizar comandos y se tiene que buscar una solución más complicada.

[4] `http://sourceforge.net/project/?group_id=612`

LENGUAJE	¿Orientado a cálculo numérico?	Nº máximo operativo de dimensiones de matrices	Velocidad de cálculo	Calidad gráficos 2D	Calidad gráficos 3D	Número funciones matemáticas disponibles	Precio corriendo sobre Linux
Mathematica	No	Infinitas	Baja	Media	Media	Muchas	Alto
MatLab	Sí	Dos	Media	Alta	Alta	Muchas	Alto
Octave	Sí	Dos	Media	Media	Pobre	Medio	Gratis
Yorick	Sí	Infinitas	Alta	Media	Media	Pocas	Gratis
PDL	Sí	Infinitas	Alta	Alta	Media	Medio	Gratis

Tabla 1: Tabla comparativa de PDL con otros lenguajes de cálculo numérico. Veáse el texto para más detalles sobre el significado de las columnas.

Para tener una referencia con otros lenguajes de programación de cálculo numérico se detalla en la Tabla 1. En dicha tabla la comparativa se realiza sobre las siguientes características:

- ¿Orientado a cálculo numérico? Esta columna hace referencia a si el lenguaje ha sido diseñado para realizar cálculo numérico de forma sencilla.
- Número máximo operativo de dimensiones de matrices. Esta columna refleja el hecho de si el lenguaje ha sido diseñado para que funcione bien, de forma operativa, con matrices de dimensión dos o más.
- Velocidad de cálculo. Una visión muy aproximada de la velocidad de cálculo del lenguaje, siempre desde el punto de vista de un lenguaje de alto nivel (mucho más lento que un programa en C). Esta velocidad está determinada con programas que tienden a favorecer los lenguajes con matrices de más de dos dimensiones. Habitualmente, por ejemplo, introducir un bucle explícitamente en Octave o MatLab supone una penalización en velocidad muy grande. Este tipo de bucles suelen ser innecesarios en lenguajes que admiten matrices de más dimensiones (PDL y Yorick).
- Calidad gráficos 2D. Calidad final de las gráficas bidimensionales generadas con ese lenguaje. Alta significa que tiene calidad de publicación.
- Calidad gráficos 3D. Calidad final de las gráficas tridimensionales generadas con ese lenguaje. Alta significa que tiene calidad de publicación.
- Número funciones matemáticas disponibles. Cada lenguaje dispone de más o menos funciones matemáticas, así como subrutinas de cálculo tales como minimización, resolución de ecuaciones diferenciales, etc.
- Precio corriendo bajo Linux. Precio del lenguaje corriendo bajo Linux.

2 Tutorial de PDL

A partir de ahora mostraremos un brevísimo tutorial para familiarizarse con PDL, utilizándolo desde la línea de comandos. Téngase en cuenta que, por motivos de simplicidad, no se contarán los aspectos "avanzados" del lenguaje, por lo cual el tutorial aquí presentado no mostrará todos los aspectos sobresalientes del lenguaje.

PDL se puede utilizar bajo la línea de comandos o dentro de un "script" Perl. Para utilizar PDL en un "script" Perl basta con introducir el comando

```
use PDL;
```

al comienzo del "script" Perl. Para más detalles consúltese el manual de PDL "Beginning PDL" mencionado en la sección "Más ayuda" 10.

Para utilizar PDL bajo la línea de comandos hay que invocarlo con el comando

```
perldl
```

Si lo tenemos instalado se ejecutará y aparecerá algo como lo siguiente

```
perlDL shell v1.30
 PDL comes with ABSOLUTELY NO WARRANTY. For details, see the file
 'COPYING' in the PDL distribution. This is free software and you
 are welcome to redistribute it under certain conditions, see
 the same file for details.
ReadLines enabled
Reading PDL/default.perldlrc...
Found docs database /usr/lib/perl5/PDL/pdldoc.db
Type 'help' for online help
Type 'demo' for online demos
Loaded PDL v2.0
perldl>
```

Nótese que en este artículo no se describe cómo instalar PDL. Para ello remítase a la documentación de su distribución para instalar el paquete correspondiente.

Los ejemplos que aparecen más abajo están descritos en forma de tutorial de forma que una línea precedida de

```
perldl>
```

son líneas que debes teclear como ejercicio. Las líneas que no comiencen con este "prompt" son las salidas del programa, NO teclees estas líneas en la línea de comandos de PDL.

Por ejemplo, en el caso

```
perldl> p 3
3
```

debes escribir `p 3` y obtienes como resultado `3`.

3 Creando PDLs

Los PDLs o "piddles" son los ladrillos básicos de PDL. Constituyen escalares de Perl pero contienen mucha más información que sólo un valor numérico. Representan un vector, una matriz o cualquier matriz n–dimensional. En esta sección veremos cómo crearlos de diferentes formas.

3.1 "Piddles" sencillos

Para crear un "piddle" se puede usar el comando `pdl`. Este comando acepta un escalar de Perl, una referencia a un "array" o un "array". Por ejemplo, para crear un vector o un "piddle" unidimensional que tenga valores de 7 a 10 escribiremos

```
perldl> $a=pdl([7..10]);
```

Para imprimir el contenido de un PDL usa el comando `print`

```
perldl> print $a;
[7 8 9 10]
```

Para escribir menos, en la línea de comandos se puede omitir el punto y coma final y además el comando `print` se puede sustituir por la letra `p` sola. Nótese que esto no es así cuando escribimos un "script" en Perl. Así pues, la siguiente expresión es equivalente a la anterior,

```
perldl> p $a
[7 8 9 10]
```

Vamos a crear un "piddle" con valores cualesquiera

```
perldl> $b=pdl([4,2,8,1,6])
perldl> p $b
[4 2 8 1 6]
```

Vamos a crear ahora uno bidimensional

```
perldl> $b=pdl([ [3,4,5], [2,8,3] ])
perldl> p $b

[
 [3 4 5]
 [2 8 3]
]
```

3.2 "Piddles" más complejos

El comando `sequence` crea un vector con una secuencia de valores. Acepta como argumentos un "array" especificando la longitud de cada una de las dimensiones del "piddle". Por ejemplo, esta línea crea un vector con valores desde 0 a 15.

```
perldl> $d=sequence(15);
perldl> p $d
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
```

Para crear una matriz 3×4

```
perldl> $m=sequence(3,4);
perldl> p $m

[
 [ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
]
```

El comando `zeroes` crea un "piddle" relleno de ceros con las dimensiones dadas por sus argumentos.

Para crear una matriz 6×7 escribiremos

```
perldl> $r=zeroes(6,7);
perldl> p $r

[
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
]
```

Y si queremos un "piddle" de cualquier dimensión con valores ascendentes en el eje x ? Para ello es útil el comando `xvals`, el cual acepta como argumentos las longitudes de cada una de las dimensiones que queremos obtener

```
perldl> $x=xvals(3,2);
perldl> p $x
 [
  [0 1 2]
  [0 1 2]
 ]
```

Un comando similar existe para recorrer el eje y

```
perldl> $y=yvals(3,2);
perldl> p $y
 [
  [0 0 0]
  [1 1 1]
 ]
```

Los comandos mencionados recorren valores enteros que comienzan en cero. Pero a veces queremos más flexibilidad y queremos comenzar en un valor x determinado y acabar en otro. Esto se puede conseguir con el comando `xlinvals`

```
perldl> $a=zeroes(3,2);
perldl> $x=$a->xlinvals(0.5,1.5);
perldl> p $x
 [
  [0.5   1 1.5]
  [0.5   1 1.5]
 ]
```

Nótese que este comando acepta tres argumentos, un "piddle" con dimensiones idénticas al "piddle" que queremos crear, un valor de inicio y un valor final. En esta última expresión se ha utilizado la notación orientada a objetos de Perl, que en este caso hace el comando más legible. Esta expresión es totalmente equivalente a haber colocado la variable que aparece antes del símbolo `->` como primer argumento de la función. Así pues una expresión equivalente sería

```
perldl> $a=zeroes(3,2);
perldl> $x=xlinvals($a,0.5,1.5);
perldl> p $x
 [
  [0.5   1 1.5]
  [0.5   1 1.5]
 ]
```

En lo que sigue utilizaremos la notación que parezca más intuitiva según el comando que utilizemos indistintamente.

Existe también un comando análogo para los valores del eje y

```
perldl> $a=zeroes(2,3)
perldl> $y=$a->ylinvals(0.3,1.3);
perldl> p $y
 [
  [0.3 0.3]
  [0.8 0.8]
  [1.3 1.3]
 ]
```

Podemos anidar varias funciones unas dentro de otras de la siguiente forma

```
perldl> $y=ylinvals(zeroes(2,3),0.3,1.3);
perldl> p $y
 [
  [0.3 0.3]
  [0.8 0.8]
  [1.3 1.3]
 ]
```

Usando la notación orientada a objetos podemos concatenar varios comandos con el símbolo `->` y obtendremos una expresión totalmente equivalente a la anterior

```
perldl> $y=zeroes(2,3)->ylinvals(0.3,1.3);
perldl> p $y
 [
  [0.3 0.3]
  [0.8 0.8]
  [1.3 1.3]
 ]
```

De nuevo, cualquier notación será utilizada indistintamente.

Los comandos `xlinvals` y `ylinvals` son útiles para crear "piddles" que será utilizados luego como entrada de los ejes x y y para representarlos gráficamente o como entrada a una función dada.

El siguiente ejemplo calcula los valores de una función gaussiana bidimensional

```
perldl> $x=zeroes(20,20)->xlinvals(-0.5,0.5);
perldl> $y=zeroes(20,20)->ylinvals(-0.5,0.5);
perldl> $gaus=exp( -($x**2)/0.05 - ($y**2)/0.02 );
```

También podemos crear números aleatorios entre cero y uno dentro de un "piddle" de dimensiones dadas con el comando `random`.

```
perldl> $r=random(2,3)
perldl> p $r
[
 [0.84018772 0.39438293]
 [0.78309922 0.79844003]
 [0.91164736 0.19755137]
]
```

De forma natural, cuando asignamos un contenido a una variable de PDL con el símbolo igual, `=` , no siempre se crea una copia nueva de los datos, sino que a menudo utiliza la mismas posiciones de memoria que la variable original. Para realmente crear una copia nueva e independiente de un "piddle" hay que usar el comando `copy` tal y como aparece en el siguiente ejemplo

```
perldl> $a=pdl[1,2,3]
perldl> $b=$a->copy
perldl> p $a
[1 2 3]
perldl> p $b
[1 2 3]
```

4 Aritmética

PDL acepta varios operadores y funciones conocidas. Los siguientes símbolos suman, restan multiplican y dividen varios "piddles".

```
+ - * /
```

Con estos operadores, al igual que el resto en este artículo, PDL actúa elemento a elemento. Por ejemplo, si multiplicamos dos PDLs del mismo tamaño y dimensiones obtenemos un PDL equivalente con cada uno de los elementos siendo el producto de cada uno de los elementos de los otros dos.

```
perldl> $a=pdl([1,2])
perldl> $b=pdl([3,4])
perldl> p $a*$b

[3 8]
```

Podemos por ejemplo calcular los puntos de una parábola.

```
perldl> $x=zeroes(5)->xlinvals(-2,2);
perldl> $y=$x**2;
perldl> p $x
[-2 -1 0 1 2]
perldl> p $y
[4 1 0 1 4]
```

El operador de exponenciación es igual que el de FORTRAN `**`.

Podemos también realizar una operación con PDLs o números de diferentes dimensiones. Por ejemplo,

```
perldl> $a=pdl[0,1,2,3]
perldl> p $a
[0 1 2 3]
perldl> $b=2*$a
perldl> p $b
[0 2 4 6]
```

En este caso estamos multiplicando un "piddle" de dimensión uno y longitud cuatro por un número. PDL lo que hace es "expandir" el número hasta abarcar el número de dimensiones del PDL de mayor tamaño. Y devuelve un "piddle" de igual tamaño que el original pero multiplicando elemento a elemento su contenido original por el número. Aquí radica la potencia de PDL, ya que no sólo tiene reglas "lógicas" para vectores y matrices, sino para todo tipo de matrices n–dimensionales. Para más información en este tema leáse la página de manual con el siguiente comando `help Indexing`.

Los operadores relaciones y lógicos son muy similares de los de C

```
> < >= <= == != !
```

Nótese que estos operadores actúan elemento a elemento. Es decir, si un "piddle" se compara con un número, el resultado es un "piddle" del mismo tamaño que el original.

```
perldl> $a=pdl([0,1,2,3,4])
perldl> p $a > 2
[0 0 0 1 1]
```

Las funciones matemáticas también actúan elemento a elemento, algunas de ellas son las siguientes

```
sin log abs atan2 sqrt cos exp
```

5 Gráficas sencillas

Para poder representar gráficamente lo que vamos creando es interesante introducir unos pocos comandos gráficos. Más adelante veremos cómo conocer comandos gráficos más complejos.

5.1 Gráficos 3D

Los gráficos tridimensionales no se cargan por defecto al arrancar PDL, con lo cual debemos cargar el módulo siguiente

```
perldl> use PDL::Graphics::TriD;
```

Para representar funciones 3D usaremos el comando `imag3d`. Podemos por ejemplo pintar la gaussiana mencionada más arriba (recordemos que antes hay que cargar el módulo 3D con la línea `use` mencionada más arriba).

```
perldl> $x=zeroes(20,20)->xlinvals(-0.5,0.5);
perldl> $y=zeroes(20,20)->ylinvals(-0.5,0.5);
perldl> $gaus=exp( -($x**2)/0.05 - ($y**2)/0.02 );
perldl> imag3d([$gaus])
```

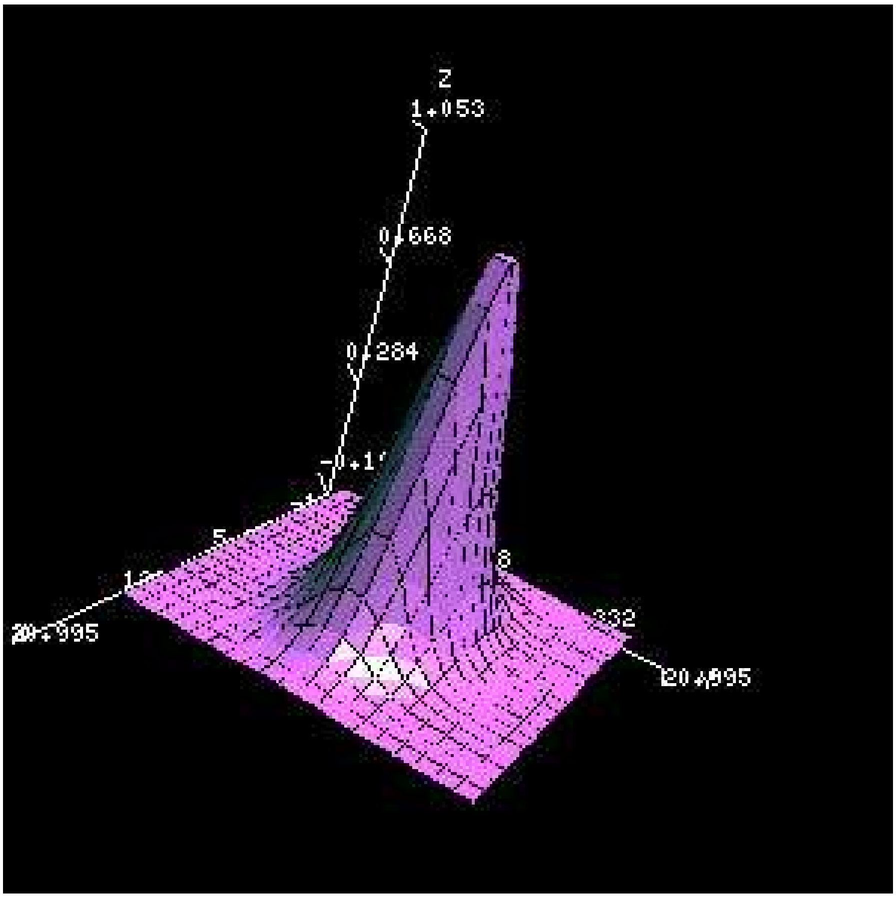

Figura 1: Función Gaussiana tridimensional

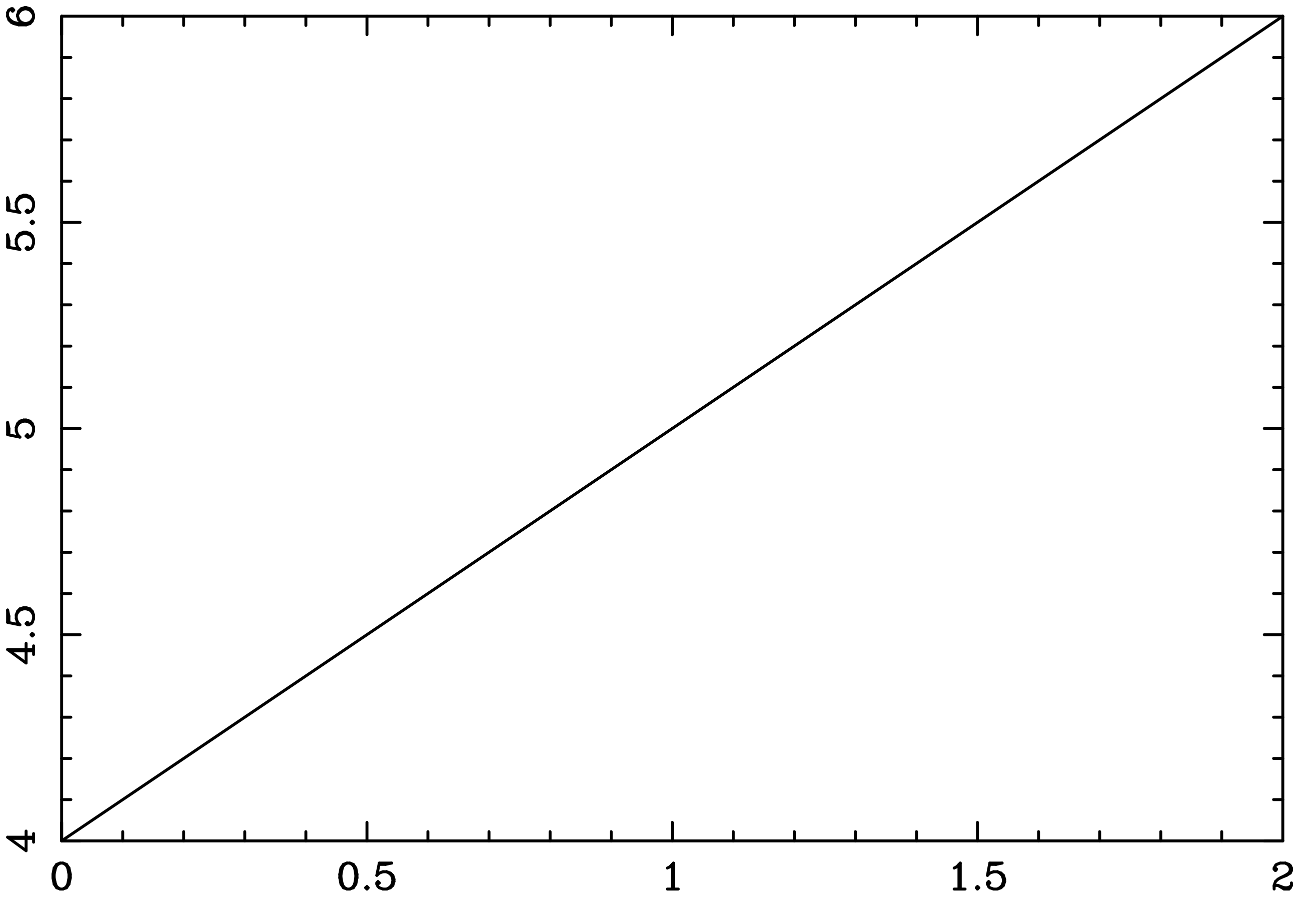

Figura 2: Gráfica de una línea

Debería aparecer algo como la Fig. 1.

Nótese que no nos devuelve el "prompt". La razón de ello es que podemos modificar la función representada en el lugar. Si ponemos el ratón sobre la gráfica podemos girarla apretando el botón izquierdo y arrastrando el ratón. Se puede tambíen hacer un zoom hacia dentro o hacia fuera con el botón derecho del ratón y arrastrándolo. Para salir de este estado pulse la tecla `q`. Volverá el "prompt". Después de ello conviene no matar la ventana gráfica para poder seguir con la sesión interactiva de PDL.

6 Gráficos 2D

Para representar una función 2D podemos usar el comando `line`. El siguiente ejemplo dibujará una línea

```
perldl> $x=pdl([0,1,2])
perldl> $y=pdl([4,5,6])
perldl> line($x,$y)
```

Nos aparecerá una pregunta preguntándonos sobre qué tipo de dispositivo queremos dibujar. Para ver todos los dispositivos disponibles teclee una `?` y pulse `<ENTER>`. Si sólo queremos que dibuje en las ventanas X escribe `/xserve`. Debería salir una bonita gráfica como la de la Fig. 2.

Podemos por ejemplo representar la parábola que creamos anteriormente tal como aparece en la Fig. 3.

```
perldl> $x=zeroes(30)->xlinvals(-1,1);
perldl> $y=$x**2;
perldl> line($x,$y)
```

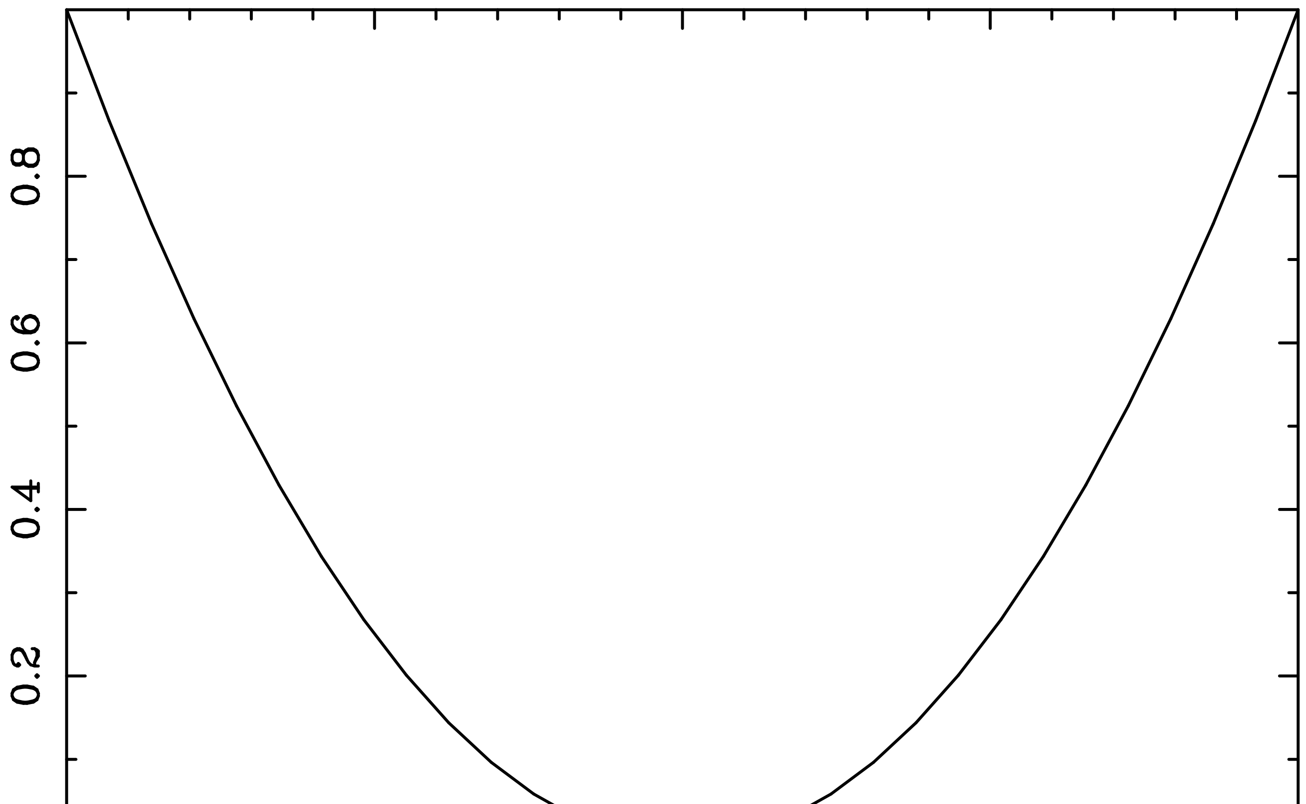

Figura 3: Gráfica de una parábola

7 Operaciones con matrices

Para ilustrar un poco lo que se puede realizar con PDL veremos un ejemplo de cómo utilizar matrices.

Para multiplicar matrices podemos utilizar la notación siguiente

```
perldl> $a=pdl[ [1,2],[3,4] ]
perldl> $b=pdl[[5,6]]
perldl> $c=$a x $b
perldl> p $c
[
 [15 18]
 [35 42]
]
```

Para calcular la inversa necesitamos cargar previamente el módulo `Slatec` y utilizar el comando `matinv` de la siguiente forma

```
perldl> use PDL::Slatec
perldl> $inv=matinv($a)
perldl> p $inv
[
 [  -2    1]
 [ 1.5 -0.5]
]
perldl> p $inv x $a
[
 [1 0]
 [0 1]
]
```

8 Análisis de Fourier

Como ejemplo ilustrativo adicional vamos a realizar un análisis de Fourier de una señal y filtrarla.

Primero crearemos la señal a procesar y la dibujaremos tal como aparece en la Fig. 4.

```
perldl> $x=zeroes(1000)->xlinvals(-20,20)
perldl> $y=sin($x/0.1)+cos($x/0.5)
perldl> line($x,$y)
```

El comando para realizar transformadas de Fourier es el `fft`, pero antes de utilizarlo hay que cargar el módulo `PDL::FFT`. Por otro lado el comando `fft` acepta como argumentos la parte real de la señal y su imaginaria y las modifica en el sitio, es decir, los "piddles" de salida son los mismos que los de entrada pero modificados. Para conservar la señal original haremos primero una copia de ella como parte real y parte imaginaria.

```
perldl> $ry=$y->copy
perldl> $iy=zeroes(1000)
```

Ahora realizamos la transformada de Fourier

```
perldl> use PDL::FFT
perldl> fft($ry,$iy)
```

Podemos ahora dibujar el espectro de potencias de dicha transformada

```
perldl> line($ry**2+$iy**2)
```

y obtenemos la Fig. 5

Creemos un filtro de forma que anulemos las frecuencias de una de las sinusoides tal y como aparece en la Fig. 5.

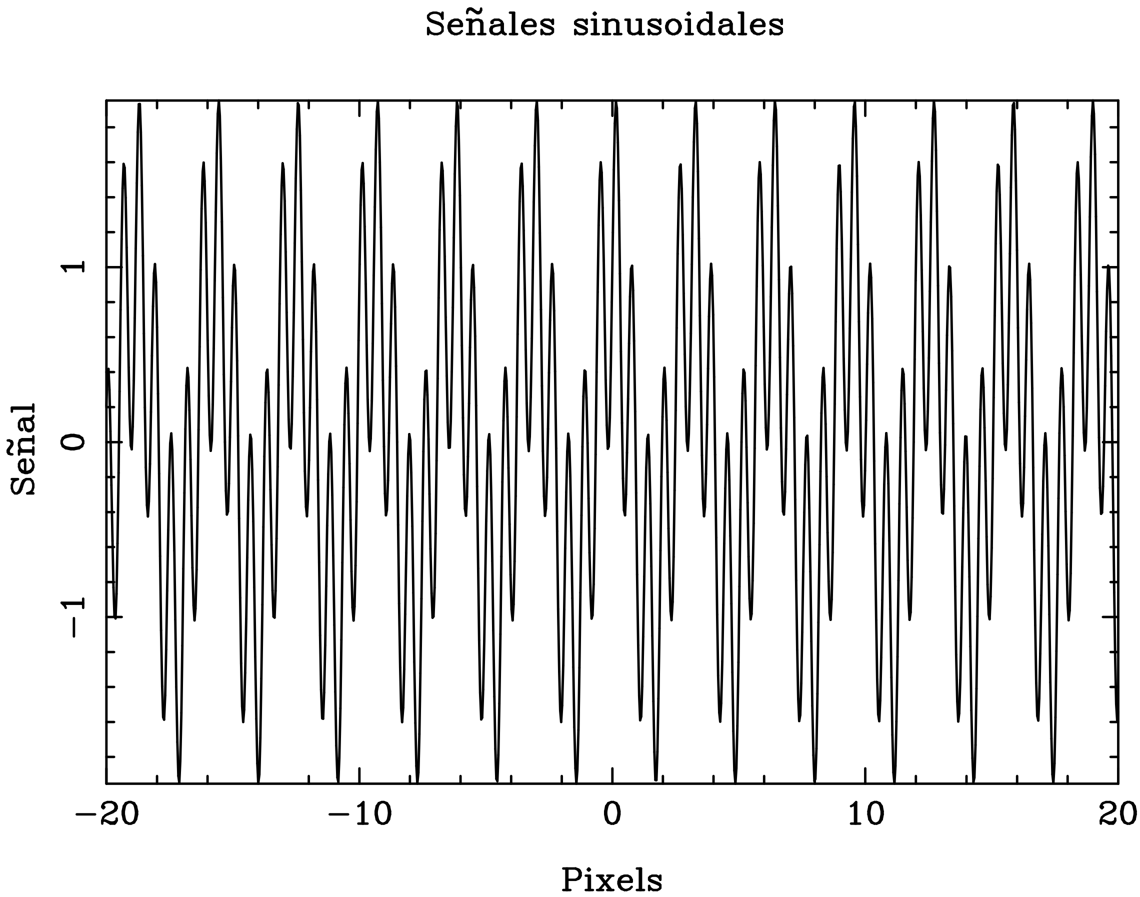

Figura 4: Señal compuesta de un seno más un coseno

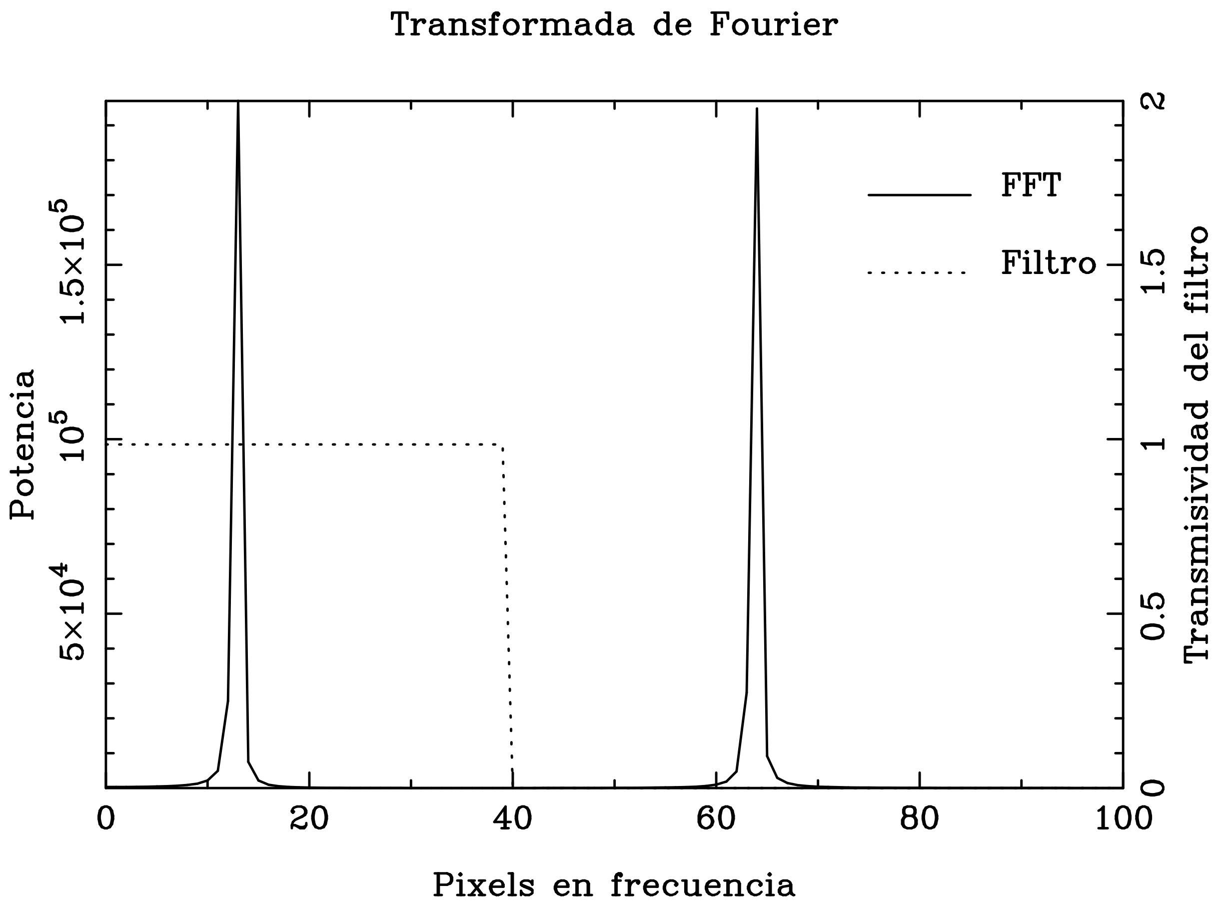

Figura 5: Transformada de Fourier y filtro aplicado

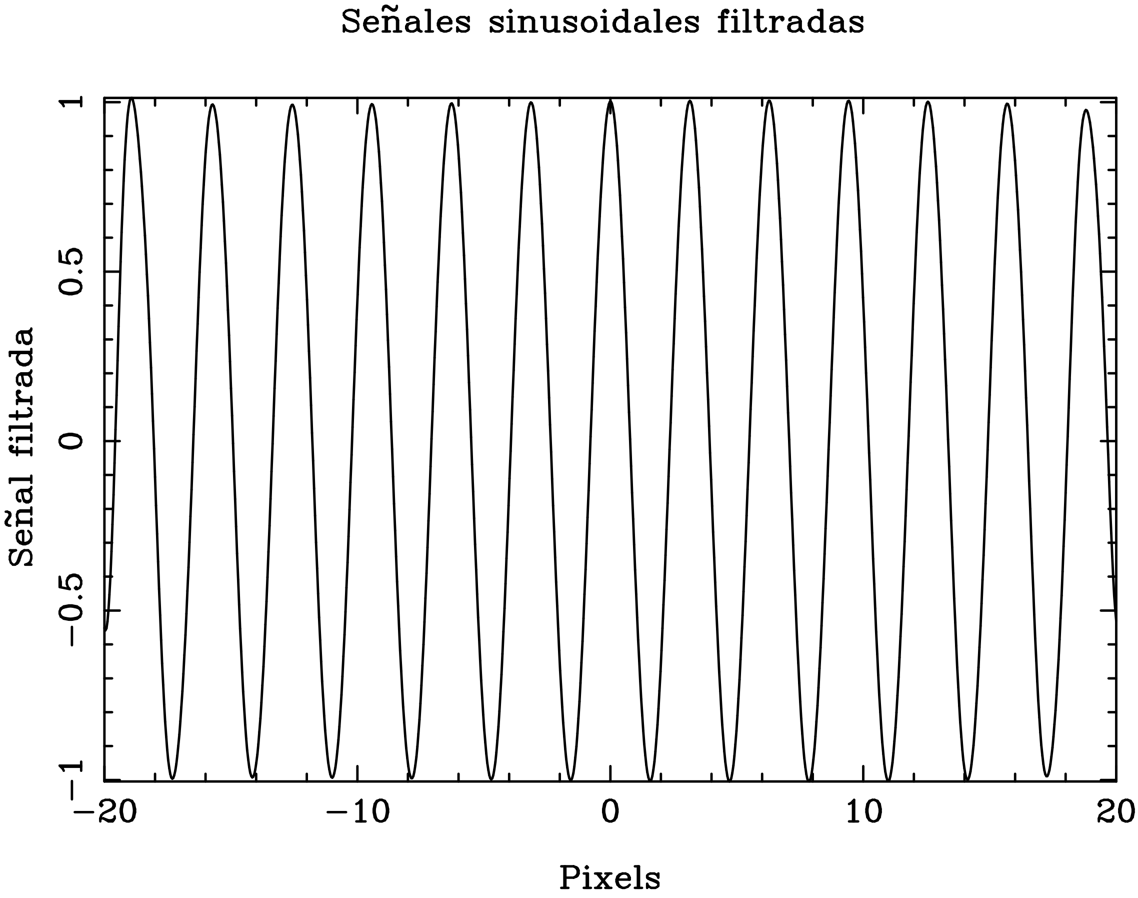

Figura 6: Señal filtrada

```
perldl> $filtro=sequence(1000)
perldl> $filtro=($filtro<40)+($filtro>960)
```

Multiplicando el filtro por las dos componentes de la transformada obtenemos la señal filtrada

```
perldl> $ry=$ry*$filtro
perldl> $iy=$iy*$filtro
perldl> line($ry**2+$iy**2)
```

Deshaciendo ahora la transformada y dibujando el resultado obtenemos la señal filtrada de la Fig. 6.

```
perldl> ifft($ry,$iy)
perldl> line($x,$ry)
```

9 Más gráficas

La parte gráfica es uno de los puntos fuertes de PDL. Existen unas demostraciones muy ilustrativas que vienen con PDL.

Para ver las demostraciones que hay escribe `demo`. Si quieres correr las demostraciones gráficas escribe `demo pgplot`, `demo 3d` ó `demo 3d2` .

Cada una de las demostraciones muestra los comandos utilizados para obtener los gráficos que se van mostrando en la terminal desde la que se invoca.

10 Más ayuda

Si te ha interesado este lenguaje y quieres saber más, puedes empezar por el manual en inglés para principiantes escrito por el autor del presente artículo "Beginning PDL" el cual estará disponible en breve en la página Web de PDL[5]

Desde la línea de comandos puedes encontrar más ayuda con el comando `help`. Para saber cómo usarlo haz

```
perldl> help help
```

Para obtener la página de manual de un comando se hace con `help nombre_comando`. También se puede usar el comando `apropos` de la forma `apropos palabra_clave` para encontrar un listado de los comandos relacionados con dicha palabra clave. Para listar los módulos disponibles haz `apropos module`. Para obtener ayuda de un módulo se hace con `help nombre_module`. Para listar los manuales disponibles en línea escribe `apropos manual`. Para obtener ayuda de un manual escribe `help nombre_manual`. Para listar todos los comandos haz `apropos .` .

Existen también varias páginas HTML disponibles en el lugar donde esté instalado PDL, normalmente en `/usr/lib/perl5/PDL/HtmlDocs`

11 Bibliografía complementaria

- "Beginning PDL", X. Calbet. Libro de introducción a PDL escrito por el autor. Se puede encontrar en la página Web de PDL[6].
- "Learning Perl", R.L. Schwartz y T. Christiansen, O'Reilly and Associates. Libro de introducción a Perl, para aprender Perl rápidamente y de forma sencilla.
- Página Web principal de Perl[7]. Página donde viene todo sobre Perl.
- Revista "on–line" sobre Perl[8]. Revista mensual sobre Perl en general donde vienen trucos de interés.

[5] http://sourceforge.net/project/?group_id=612
[6] http://sourceforge.net/project/?group_id=612
[7] http://www.perl.com
[8] http://www.perlmonth.com