GNU/Linux y la Simulación Numérica de Ecuaciones en Derivadas Parciales. Un Modelo en Oceanografía

J. Rafael Rodríguez Galván*

Departamento de Matemáticas. Universidad de Cádiz.

rafael.rodriguez@uca.es

Agosto 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

En este trabajo se presenta una panorámica general del uso del sistema operativo GNU/Linux y de las herramientas de software libre para el desarrollo de proyectos científicos basados en la simulación numérica de ecuaciones en derivadas parciales.

Tomando como referencia la experiencia del autor en la resoución numérica de un modelo matemático en Oceanografía, se comentan todos los aspectos relacionados con lo anterior: programación y validación del código numérico, post-proceso de resultados y generación de informes y comunicaciones.

1 Introducción

La aproximación numérica de ecuaciones en derivadas parciales constituye una rama fundamental del Análisis Numérico y un campo de investigación en constante actividad. Es muy grande el número de fenómenos que son modelados mediante ecuaciones de este tipo, provenientes de campos tan diversos como la Física, la Ingeniería, la Oceanografía, la Economía, etc.

En los casos más realistas, no es posible calcular de forma exacta la solución de este tipo de ecuaciones, siendo necesario recurrir a técnicas numéricas que, explotando aquellas propiedades que hayan sido previamente deducidas de forma

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

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

teórica, nos permitan desarrollar algoritmos mediante los cuales sea posible obtener, de forma aproximada, un conjunto finito de datos que representen los valores que toman las soluciones en un dominio discreto.

El llevar a la práctica lo anterior significa realizar programas de ordenador que, a partir de unos datos conocidos, como la situación inicial de las magnitudes estudiadas y la evolución de las mismas en la frontera del dominio, devuelvan las soluciones, en la forma de conjuntos discretos de datos que representan los valores que toman dichas magnitudes. Además de esto, suelen ser necesarias tanto una etapa previa de pre-proceso de los datos iniciales y de contorno como una etapa final de post-proceso y visualización gráfica de los valores obtenidos como solución de las ecuaciones.

Los programas asociados a la resolución numérica de ecuaciones en derivadas parciales requieren de una gran potencia de cálculo, tanto mayor cuanto más realistas pretenden ser los resultados. Por este motivo, muchos de los grupos de investigación universitarios que trabajaban en este campo optaron por realizar su trabajo en estaciones de trabajo comerciales de tipo UNIX, lo cual significaba importantes desembolsos económicos. A medida que se ha ido produciendo la madurez de los entornos GNU/Linux, éstos se han desvelado como una alternativa natural: bajo este sistema operativo, un simple ordenador personal se convierte en una estación de trabajo flexible y estable, con características de multitarea y multiusuario y con una excelente relación calidad/precio. Además, para requerimientos de cálculo más exigentes, se puede optar por equipos más potentes, como ordenadores personales con varios procesadores, otras arquitecturas *hardware* (por ejemplo, sistemas *Alpha*) e incluso *clusters* de ordenadores. Por supuesto, se siguen utilizando equipos comerciales de tipo UNIX e incluso centros de cálculo dotados de grandes ordenadores para trabajos con requerimientos de cálculo realmente exigentes.

Es necesario, además, tener en cuenta el siguiente hecho: una distribución de GNU/Linux, como puede ser Debian, Red Hat, etc. es mucho más que un sistema operativo muy avanzado: constituye una completa caja de herramientas mediante las cuales es posible afrontar todos los pasos relacionados con la simulación numérica, partiendo del diseño y programación del código, pasando por la realización de experimentos que lo validaen y terminando por la generación de documentos para la exposición de resultados. Además, muchas de las herramientas, *software* libre como el compilador de C++ *egcs* o el lenguaje para la composición de documentos LaTeX, se encuentran entre las más avanzadas en su género.

En las siguientes secciones, se tratará de ilustrar todo lo anterior, tomando como base un modelo de Oceanografía en el que se encuentra actualmente trabajando el autor, en colaboración con el profesor F. Guillén, del departamento de Ecuaciones Diferenciales y Análisis Numérico de la Universidad de Sevilla.

2 Un Modelo Matemático en Oceanografía

Las ecuaciones primitivas [13] modelan el comportamiento de algunos fluidos geofísicos (atmósfera, océanos, etc.), en donde se tienen dominios cuya profundidad es pequeña en relación con las dimensiones horizontales.

Formalmente, se obtienen de las ecuaciones de Navier-Stokes imponiendo algunas simplificaciones: presión hidrostática (dependiente de forma lineal de la

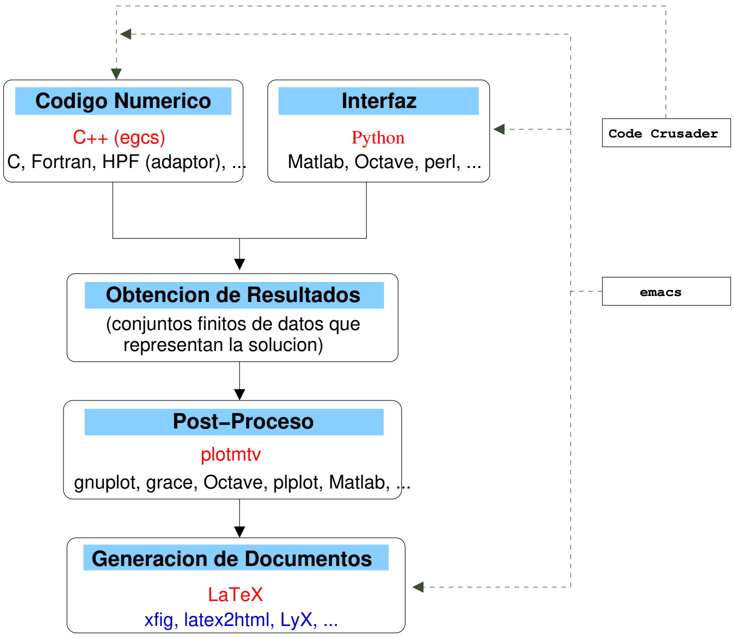

Figura 1: Pasos en simulación numérica, señalando en cada caso las aplicaciones empleadas en nuestro modelo en oceanografía (en rojo) y posibles alternativas

profundidad) y “techo rígido” (la superficie del fluido no presenta movimientos verticales). Por simplicidad, suponemos densidad constante y que los efectos debidos a la temperatura y salinidad se pueden desacoplar del flujo dinámico.

Aunque el esquema que presentamos es generalizable al caso 3-dimensional, nos centraremos aquí en la simulación numérica de las ecuaciones primitivas 2-dimensionales:

$$
\textit{(EP-2D)}\left\{
\begin{array}{rcll}
\partial_t u + u\partial_x u + v\partial_z u - \nu_h \partial^2_{xx} u - \nu_v \partial^2_{zz} u & & & \\
+ \partial_x p_s & = & f & \text{en} \quad \Omega\times(0,T), \\
v(t;x,z) & = & \displaystyle\int_z^0 \partial_x u(t;x,s)\,ds & \text{en} \quad \Omega\times(0,T), \\
\displaystyle\int_{-D(x)}^0 u(x,z)\,dz & = & 0 & \text{en} \quad \omega\times(0,T), \\
\nu_v \partial_z u|_{\Gamma_s} = g_s, \quad u|_{\Gamma_b} = 0, \quad u|_{t=0} = u_{1,0} & & & \text{en} \quad \Omega,
\end{array}
\right.
$$

donde $\Omega = \{(x,z) \;/\; x\in\omega,\; -D(x) < z < 0\}$ es el dominio 2D ocupado por el fluido (figura 2), siendo $\omega \subset I\!R$ un intervalo abierto y $D:\overline{\omega} \longrightarrow I\!R$ la función profundidad. Los siguientes parámetros se suponen conocidos, y en función de sus valores obtendremos diferentes comportamientos de las soluciones: ν_h y ν_v son los coeficientes de viscosidad horizontal y vertical, f son fuerzas externas (dependientes por ejemplo, de la salinidad y temperatura) y en g_s se encierran los efectos de fricción del viento sobre la superficie. Las incógnitas del problema son las funciones $u, v : \Omega\times(0,T) \longrightarrow I\!R$ (velocidad horizontal y vertical del flujo, respectivamente) y $p_s : \omega\times(0,T) \longrightarrow I\!R$ (presión superficial).

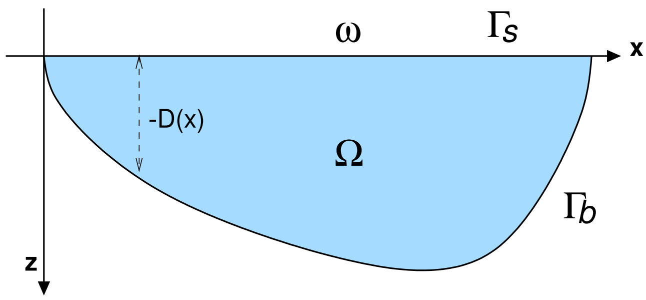

Figura 2: El dominio, Ω

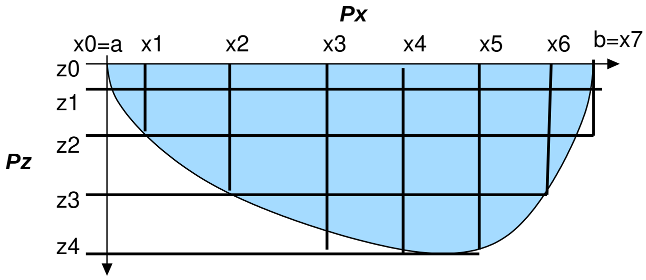

Figura 3: Discretización del dominio

Las ecuaciones anteriores presentan serias dificultades desde el punto de vista numérico: no linealidad, evolución con el tiempo, acoplamiento de las incógnitas,... El método que estamos empleando para su resolución se basa en un esquema fraccionario de proyección en tiempo [9] junto con una discretización espacial usando productos de elementos finitos. A continuación, comentaremos este método de forma muy breve; para más detalles, ver [10]

Para la discretización en la variable temporal, t, de las ecuaciones, fijamos un paso de tiempo, Δt, y construimos la sucesión de instantes $t_0 = 0, t_1 = \Delta t, t_2 = 2\Delta t, \ldots, t_k = k\Delta t, \ldots$ Tratamos de calcular, para cada etapa de tiempo $k > 0$, funciones u^k, v^k y p_s^k que aproximen, en el instante t_k, a las velocidades u, v y a la presión superficial, p_s, respectivamente. Para ello, y con el fin de separar las dificultades, se ha empleado un esquema fraccionario en tiempo, de proyección lineal (o semi-implícito) en tres pasos.

El primer paso y el más complejo, consiste en hallar una primera predicción de la velocidad horizontal, como solución de un sistema lineal más sencillo que *(EP-2D)*. En un segundo paso, corregimos la predicción realizada para hallar la aproximación definitiva de la velocidad horizontal y calculamos una estimación de la presión superficial. En el tercer y último paso, obtenemos la velocidad vertical.

Para la discretización en espacio, aproximamos el dominio, Ω, que ocupa el fluido de la siguiente forma (figura 3): fijamos una partición $\mathcal{P}_x = \{a = x_0 < x_1 < \ldots < x_N = b\}$ de la superficie $\omega = [a, b]$. Por otra parte, consideramos M capas de profundidad, definidas a través de una partición

$\mathcal{P}_z = \{0 = z_0 > z_1 > \ldots > z_M = -D_{max}\}$, siendo $D_{max} = \max\limits_{x\in\omega} D(x)$. De esta forma, cada z_n representa la profundidad del nivel horizontal n ($n = 0, \ldots, M$). El producto cartesiano de ambos mallados define, como se aprecia en la figura 3, una discretización del dominio. Se ha utilizado el Método de los Elementos Finitos definido mediante el producto de funciones base polinómicas en la variable x (definidas en el mallado de la superficie del dominio) por funciones base sobre polinómicas en z (definidas en el mallado en profundidad). Utilizando este método calculamos, en cada etapa de tiempo, las soluciones aproximadas.

El esquema empleado es generalizable al caso 3-dimensional, sin más que sustituir las funciones polinomicas en x, definidas sobre la superficie unidimensional del fluido, por funciones en x, y, definidas sobre la superficie bidimensional.

3 Programación en GNU/Linux del código de simulación numérica

Una vez discretizadas las ecuaciones en derivadas parciales, debemos pasar a su resolución efectiva. Para ello, elaboramos un programa de ordenador que, a partir de los datos iniciales (en el instante t=0) y conociendo los datos de contorno (en nuestro caso, los valores que toman las incógnitas en las paredes y en el fondo del recinto), genere, en cada instante de tiempo $t_k = k \cdot \Delta t$, los vectores $\mathbf{u}^k$, $\mathbf{v}^k$ y $\mathbf{p}_s^k$ que representen respectivamente, el campo de velocidades horizontal y vertical y la presión superficial en el instante t_k.

En el caso 2-dimensional, hemos hallado algunas propiedades que simplificarán algo el trabajo. Por un lado, podemos evitar el cálculo de la presión superficial, variable que, en principio, no es de interés físico. Por otra parte, es posible calcular de forma explícita las velocidades verticales. De esta forma, en cada iteración de tiempo se trata, básicamente, de resolver un sistema lineal de ecuaciones $A\mathbf{u}^k = \mathbf{b}$, proveniente del método de elementos finitos. El formato que se escogió para los elementos finitos hace que la matriz A sea tridiagonal por bloques, los cuales son, a su vez, matrices tridiagonales. En el segundo miembro, $\mathbf{b}$, se encierran datos conocidos, como el efecto de fricción del viento sobre la superficie del fluido.

Para elaborar el programa, se ha elegido el *lenguaje de programación C++*, utilizando *egcs* [5], compilador libre (bajo licencia GNU-GPL) con soporte de ANSI C++, cuyas características lo hacen estar al nivel de los más potentes compiladores comerciales. Por supuesto, si lo hubiésemos considerado adecuado, también podríamos haber utilizado compiladores comerciales bajo GNU/Linux, como *KAI C++*.

A pesar de seguir siendo Fortran el lenguaje más extendido en el campo de la simulación numérica, nosotros decidimos optar por el uso del lenguaje de programación C++ aprovechando sus características de orientación a objetos para representar matrices, mallados de elementos finitos, etc. El uso de este lenguaje, que poco a poco parece empezar a abrirse un hueco en la programación de métodos numéricos, nos confiere mayor flexibilidad tanto en el momento de la elaboración del programa como en la reutilización del código para proyectos posteriores.

Entre las ventajas de GNU/Linux se encuentra el poder contar con compiladores de prácticamente cualquier lenguaje. Por ejemplo, la programación en Fortran es perfectamente factible, sin más que emplear el compilador GNU de Fortran 77. Desgraciadamente, este compilador no cuenta aún con soporte para Fortran 90, especificación que posee características más avanzadas que la anterior: tipos de datos compuestos, asignación dinámica de memoria, sobrecarga de operadores, etc. Existe un proyecto para crear un compilador de Fortran 95 con licencia libre (http://g95.sourceforge.net/) que todavía está en estado embrionario.

En est ámbito, una posibilidad interesante puede ser el utilizar *adaptor* [1]. Adaptor (Automatic DAta Parallelism TranslaTOR) es un sistema de compilación para HPF (High Performance Fortran) con licencia no comercial. HPF es una extensión de Fortran con la que se pueden escribir de forma sencilla programas paralelos, con paralelización orientada a datos. Estos programas podrán ejecutarse en ordenadores con varios procesadores (por ejemplo, un ordenador dual ejecutando Linux en modo SMP) o bien en *clusters* de varios ordenadores conectados en red (utilizando una librería de paso de mensajes, como PVM o MPI). Por ejemplo, en el curso de doctorado "Análisis Numérico de EDP no Lineales" que organizó en el año 1999/2000 el Departamento de Ecuaciones Diferenciales y Análisis Numérico de la Universidad de Sevilla junto con el Departamento de Informática y Análisis Numérico de la Universidad de Córdoba, se utilizó *adaptor* para realizar ejercicios de programación numérica en un cluster formado por los ordenadores de un aula de informática, los cuales se encontraban ejecutando PVM bajo Linux.

Volviendo a la resolución práctica del problema en oceanografía que estamos estudiando, el utilizar el método de los elementos finitos significa que nos encontraremos finalmente con la necesidad de resolver sistemas de ecuaciones. Estos sistemas suelen ser de grandes dimensiones (miles de ecuaciones), especialente si tratamos de estudiar casos algo realistas, en los cuales la discretización del dominio ha de ser muy fina. Para implementarlos resolverlos en la práctica, podríamos haber usado alguna de las excelentes bibliotecas matriciales que existen para C++, como *Blitz++* [2], *MTL* [15] o *SparseLib++* [24], las dos últimas acompañadas de bibliotecas de métodos iterativos para la resolución de sistemas de ecuaciones lineales. Todas estas bibliotecas tienen licencias no comerciales, y han sido verificadas con *egcs* (entre otros compiladores de C++). No obstante, con el fin de aprovechar la estructura tridiagonal de las matrices asociadas, siempre pensando en el ahorro de memoria, pues la matriz del sistema tendrá una gran dimensión, optamos por desarrollar nuestra propia biblioteca[3] de clases C++. En ella, definimos las matrices tridiagonales mediante un clase patrón:

```
template <class T> MatrizTridiag<T>;
```

en la que, para ahorar memoria, se almacenan exclusivamente los elementos diagonales, super-diagonales y sub-diagonales de la matriz. Así, podemos implementar la matriz por bloques del sistema, A, como un objeto del tipo

```
MatrizTridiag< MatrizTridiag<double> > A;
```

y, sin más que sobrecargar el operador `( )`, podemos acceder al elemento (k, l) del bloque (i, j) mediante la expresión:

[3] Probablemente, la biblioteca *MTL* también podría haber sido utilizada con este fin.

```
A(i,j)(k,l);
```

Para resolver los sistemas lineales, se utiliza un método clásico de relajación por bloques, en el que los sistemas asociados a los bloques tridiagonales se resuelven de forma explícita mediante un algoritmo de tipo *Thomas*. Para la representación de los mallados de elementos finitos, se han contruido las clases C++ adecuadas.

Como editor de texto y entorno de programación, utilizamos *emacs* [4], junto con una herramienta gráfica, *Code Crusader* [3], que facilita el acceso a los ficheros fuente de la biblioteca de clases C++, lo cual resulta muy útil a medida que el numero de clases que contiene ésta comienza a ser grande.

Por otra parte, en un programa de simulación numérica es interesante contar con una interfaz de usuario adecuada pues, una vez desarrollado, es necesario realizar numerosas pruebas que lo validen, introduciendo variaciones en los parámetros del programa o en los datos iniciales y comparando los resultados con aquellos observables en el "mundo real". En nuestro caso, se utilizó el lenguaje de programación *Python* [22], desarrollando un módulo que interactúa con el programa C++. Mediante este módulo, podemos acceder interactivamente a las funciones de bajo nivel, escritas en C++, para calcular las soluciones en etapas de tiempo sucesivas, acceder a las variables más importantes, modificar sus valores, observar los resultados, etc.

Se eligió el lenguaje *Python* para la interfaz de usuario de nuestro programa por tratarse de un lenguaje interpretado con características avanzadas y fácil de enlazar con C/C++. Además, *Python* cuenta con módulos especializados para el cálculo numérico, para la representación gráfica de datos, etc. Por supuesto, existen otras muchas posiblidades, la más extendida de las cuales puede que sea el uso del programa *Matlab* [16], lenguaje orientado al análisis matricial y que puede ser utilizado además para la representación gráfica de resultados. En nuestro caso, lo descartamos por no considerar rentable la inversión en un programa comercial, existiendo otras alternativas libres que resultaban tanto o más válidas para nuestros propósitos. Por otro lado, una posibilidad, que no descartamos usar en un futuro próximo, es usar *octave* [17], lenguaje GNU interpretado, orientado al análisis numérico y matricial, similar a *Matlab*, que puede también enlazarse con código C. Por último, se utilizó *Phyton* en vez de otro lenguaje de interpertado de características similares como *perl* simplemente por una cuestión de preferencias personales, pues este último podría haber resultado igualmente válido.

4 Post-Proceso y Representación Gráfica de los Resultados

Para la visualización gráfica de los resultados, contamos en GNU/Linux con bastantes herramientas de licencia libre. Por ejemplo, en los experimentos que presentamos aquí, hemos empleado el programa *plotmtv* [19]. Existen también otras aplicaciones, como *grace* [8], o *gnuplot* [7], que creemos nos serán de utilidad para ser empleadas en próximas etapas. Todos estos programas tienen características similares, puedendo tomar sus datos de un fichero que ha sido escrito en un formato adecuado por el programa C++ o por su interfaz *Python*, para representarlos gráficamente. Las gráficas resultantes pueden ser visualizadas en pantalla o bien exportadas hacia un fichero postscript, que posteriormente podremos mandar a la impresora o incluir en un documento LaTeX, por ejemplo. Los tipos de gráficas que pueden generar (funciones de una o dos variables, campos de velocidades, curvas de nivel, etc.) son en todos los casos muy similares, si bien existen ciertas diferencias que hacen que, a la hora de utilizarlos en un caso concreto, sea conveniente decantarse por uno de ellos. Así, los campos de velocidades que aparecen en las figuras 4, 5 y 6, han sido generados por *plotmtv*, con el que resulta sencillo hacer este tipo de gráficas de vectores. Sin embargo, este programa no cuenta con algunas características interesantes, como la posibilidad de generar animaciones, para lo cual podremos recurrir, por ejemplo, a *gnuplot*.

Existen muchas más posibilidades para la representación de datos, como el empleo de bibliotecas de funciones gráficas, como *plplot* [20], o el uso de paquetes como *octave* (que emplea, a su vez, *gnuplot* o, en las últimas versiones, la biblioteca *plplot*). Asimismo, podemos contar con *R* [23], lenguaje de tratamiento estadístico y representación de datos, o emplear lenguajes interpretados como *Python* o *perl* [18], que cuentan con módulos adaptados a la manipulación numérica de datos y a su representación gráfica. Por supuesto, siempre es posible usar aplicaciones comerciales, la más usada de las cuales puede que sea *Matlab*. Algunas de ellas, como *gid* [6] (desarrollada por profesores de la Universidad Politécnica de Cataluña), integra heramientas de pre-proceso y de post-proceso de datos a través de una interfaz gráfica (disponible tanto para sitemas UNIX como para Windows).

Siguiendo con el programa de aproximación numérica de las Ecuaciones Primitivas del océano que estamos empleando como referencia, presentaremos aquí algunos de los experimentos numéricos que hemos realizado para validar el esquema, obteniendo resultados satisfactorios. En concreto, se muestran los resultados de la aproximación del problema lineal estacionario, en el cual realizamos iteraciones en tiempo, analizando si existe algún estado final hacia el cual converjan las soluciones (velocidades y presión) a medida que el tiempo avanza.

Mostraremos aquí los resultados obtenidos en tres dominios distintos. Todos ellos tienen 30.000 metros de longitud superficial por 150 de profundidad máxima. El viento es constante hacia la derecha con velocidad máxima de 7.5 m/s y se utiliza un paso de tiempo $\Delta t = 60$ segundos.

En el primer ejemplo (figura 4) se alcanza la solución exacta (con tolerancia $\varepsilon = 10^{-6}$) después de 1579 iteraciones de tiempo (algo más de 26 horas). La velocidad horizontal máxima del fluido, alcanzada en el punto medio de la superficie, es de 0.695894 m/s. En el segundo (figura 5), se llega a la solución estacionaria tras 2589 iteraciones de tiempo (algo más de 43 horas) con velocidad horizontal máxima de 0.787584 m/s. En el tercero (figura 6), han de transcurrir 3032 iteraciones (50 horas y media) y se obtiene una velocidad horizontal máxima de 0.352651 m/s.

5 Generación de Documentos y Presentaciones

Si, después de muchos días de esfuerzos delante del ordenador, los resultados obtenidos de la simulación numérica son satisfactorios, será gratificante el presentarlos públicamente, ya sea con fines educativos, en congresos de investi-

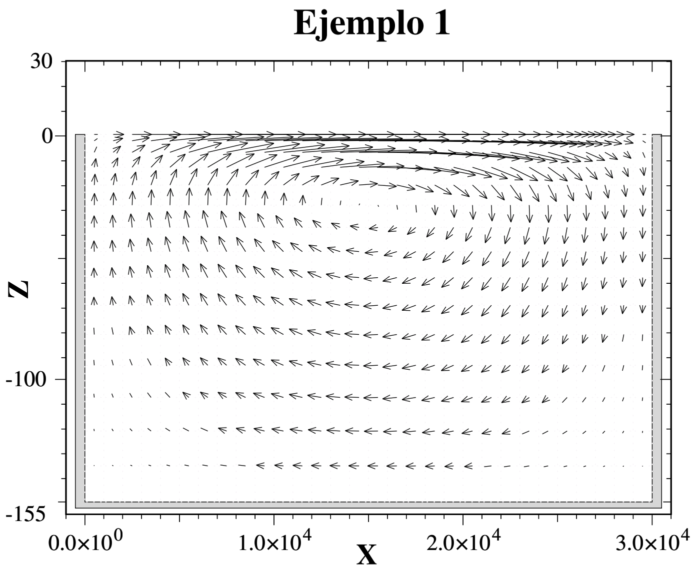

Figura 4: Solución estacionaria, ejemplo 1

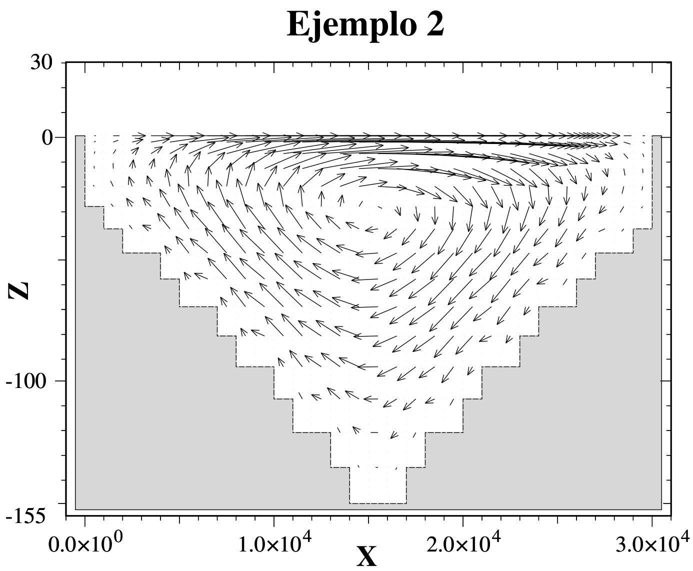

Figura 5: Solución estacionaria, ejemplo 1

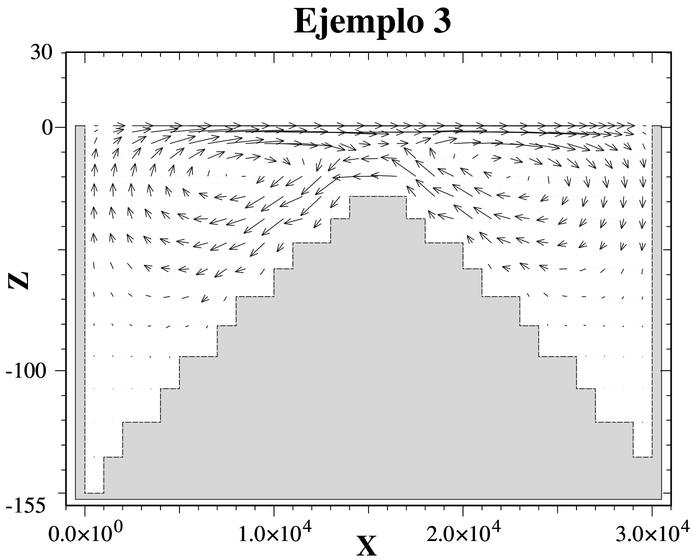

Figura 6: Solución estacionaria, ejemplo 3

gación, etc. De nuevo, podemos contar con el sistema operativo GNU/Linux y con numerosas herramientas que nos ayudarán a realizar la presentación de resultados. Obviamente, lo que sigue es de interés en un ámbito mucho más amplio que el de la simulación numérica, resultando válido para la generación de todo tipo de documentos científicos.

La clave será el uso de LaTeX, lenguaje para la generación de documentos de alta calidad y muy adecuado para trabajos matemáticos y científicos. Si bien es necesario superar la dificultad inicial del aprendizaje del lenguaje, sus características (calidad de los documentos, composición lógica, adaptación a estructuras matemáticas complejas, independencia de la plataforma, ...) hacen de él un potente instrumento. En GNU/Linux podemos contar con LaTeX, concretamente la distribución *teTeX*, desarrollada para sistemas UNIX. Además disponemos de numerosas utilidades asociadas. Por ejemplo, contaremos con un modo para emacs, *auctex*, que nos facilitará el uso de este editor de textos para la creación de documentos LaTeX. Una herramienta, *xfig* [25], para dibujar objetos gráficos de forma interactiva y exportar los resultados, usualmente a formato postscript, con el fin de ser incorporados por LaTeX(por ejemplo, las figuras 2 y 3 han sido generadas mediante *xfig*). Un entorno visual, *LyX* [14], que podremos utilizar como un procesador de texto "conveccional", pero que utiliza LaTeX para la generación final de documentos. Conversores a lenguaje HTML, herramientas para convertir el documento a formatos postscript o pdf y para post-procesar los resultados, etc

En principio, si pretendemos usar LaTeX para realizar un documento impreso, como una ponencia para un congreso, deberemos escoger para nuestro documento un estilo estándar, como *article*. Sin embargo, si necesitamos realizar transparencias para presentar los resultados, podremos elegir entre los estilos "no estándar" *foiltex* y *seminar*, ambos presentes en teTeX (la licencia del primero no es completamente libre). Si pretendemos elaborar un póster (en formato A3, A2, A1 o A0), podemos emplear el estilo *a0poster*. Este paquete no se encuentra en la distribución teTeX, pero es accesible desde internet, por ejemplo en `http://www.ucc.ie/cgi-bin/ctan`.

Por otra parte, si queremos publicar un artículo en internet, podemos contar con el programa *latex2html* [12], que convierte documentos LaTeX en formato HTML, traduciendo fórmulas y ecuaciones en gráficos de poco tamaño que pueden ser mostrados con facilidad por los navegadores de internet. Por último, si nuestra intención es generar una presentación usando el ordenador junto con un cañón proyector de vídeo, podremos usar el programa *pdflatex* para generar, a partir del fichero LaTeX, un documento *pdf*, que podremos mostrar a pantalla completa (bajo Linux o bajo cualquier otra plataforma) usando, por ejemplo, *acrobat reader*, *gv*, *xpdf* (el primero de ellos no con licencia totalmente libre). Se están desarrollando programas, como *ppower4* [21], para postprocesar los ficheros *pdf* de tal modo que se puedan incluir efectos (cortinillas, etc) a nuestro antojo en la forma de órdenes LaTeX especiales.

Con alguna frecuencia, puede ser necesario elaborar versiones del trabajo realizado en varios de los formatos que se han comentado. Por ejemplo, podemos necesitar enviar un artículo para ser publicado en las actas de a unas jornadas cientificas, a la vez que realizar una presentación del mismo, en forma de transparencias o mediante proyector de vídeo. Posteriormente, quizás nos resulte atractivo el convertirlo en formato HTML para depositarlo en en nuestra página web. El realizar por separado cada una de estas tareas conllevaría bastante esfuerzo; sin embargo, el trabajo se reduce considerablemente si creamos un artículo principal en LaTeX, quizás listo para ser publicado, a partir del cual, introduciendo posiblemente algunas modificaciones para adecuarlo al nuevo formato, crear una presentación o (usando *latex2html*) o generar un documento en formato HTML.

Por último, señalar que el lenguaje de composición de documentos LaTeX es, de hecho, un estándar para el intercambio de documentos matemáticos o científicos complejos, no existiendo, prácticamente, alternativas a su uso. Por ejemplo, el emplear editores de texto "conveccionales" como *Microsoft Word* o *Sun StarOffice*, para la generación de este tipo de documentos no solamente resulta tedioso, sino que los resultados obtenidos no alcanzan la calidad de los documentos escritos con LaTeX. Algo similar ocurre con las transparencias o, quizás, con las presentaciones mediante cañón proyector de vídeo: la utilización programas como *Microsoft PowerPoint*, bajo sistemas operativos de tipo *Windows*, es desaconsejable frente a la potencia de LaTeX para la generación de documentos matemáticos.

6 Conclusiones

Tomando como hilo conductor la experiencia del autor en la implementación de un modelo numérico en oceanografía, se han abordado diferentes cuestiones relacionadas con el uso del sistema sistema operativo GNU/Linux como entorno para la simulación numérica: programación del código, post-proceso y representación gráfica de resultados y, por último, generación de documentos y presentaciones. Para cada uno de estos pasos, se han subrayado tanto las distintas aplicaciones que hemos empleado en nuestro trabajo, como las diferentes alternativas, en muchas ocasiones igualmente válidas, con las que podemos contar, siempre haciendo énfasis en aquellas que cuentan con licencia libre.

El sistema GNU/Linux constituye una plataforma estable y potente que cuenta con herramientas muy adecuadas para la realización de proyectos relacionados con la simulación numérica de ecuaciones en derivadas parciales.

7 Agradecimientos

A Virgilio Gómez Rubio que, por convencerme de que podía ser interesante contar mi experiencia en la utilización de GNU/Linux para la simulación numérica, es el resposable último de este trabajo. Y a todos aquellos que, de forma desinteresada, se encuentran implicados en el desarrollo de Linux y de *software* libre.

Este trabajo ha sido parcialmente financiado por el proyecto PB98-0583 del Ministerio de Eduación y Cultura.

Referencias

[1] *adaptor*, `ftp://ftp.gmd.de/gmd/adaptor`

[2] *Blitz++*, `http://www.nobjects.com`

[3] *Code Crusader*, `http://www.cco.caltech.edu/~jafl/jcc`

[4] *emacs*, `http://www.gnu.org/software/emacs`

[5] *egcs (GNU C++)*, `http://gcc.gnu.org`

[6] *gid*, `http://gid.cimne.upc.es`

[7] *gnuplot*, `http://www.cs.dartmouth.edu/gnuplot_info.html`

[8] *grace*, `http://plasma-gate.weizmann.ac.il/Grace`

[9] F. Guillén González, M.V. Redondo Neble *Convergencia de algunos esquemas numéricos hacia el modelo evolutivo de Ecuaciones Primitivas.* Actas XVI CEDYA, VI CMA, University of Las Palmas de Gran Canaria, 1999, 1165-1172.

[10] F. Guillén González, R. Rodríguez Galván *Aproximación Numérica Mediante un Esquema de Proyección de las Ecuaciones Primitivas 2D* Actas II Jornadas de Análisis de Variables y Simulación Numérica del Intercambio de Masas de Agua a través del Estrecho de Gibraltar, University of Cádiz, 2000.

[11] *KAI C++*, `http://www.kai.com/C_plus_plus/`

[12] *LaTeX2html*, `http://www-texdev.mpce.mq.edu.au/l2h/docs/manual`

[13] J.L. Lions, R. Teman, S. Wang. *New formulations of the primitives equations of the atmosphere and applications.* Nonlinearity, 5 (1992), 237-288.

[14] *LyX*, `http://www.lyx.org`

[15] *MTL*, `http://www.lsc.nd.edu/research/mtl/`

[16] *Matlab*, `http://www.mathworks.com/products/matlab/index.shtml`

[17] *octave*, `http://www.che.wisc.edu/octave`

[18] *perl*, `http://www.perl.com`

[19] *plotmtv*, `ftp://ftp.x.org/contrib/applications`

[20] *plplot*, `http://www.plplot.org`)

[21] *ppower4*, ftp://ftp.dante.de/tex-archive/help/Catalogue/entries/ppower4.html

[22] *Python*, http://www.python.org

[23] *R*, http://www.r-project.org

[24] *SparseLib++*, http://math.nist.gov/sparselib++/

[25] *xfig*, http://www.xfig.org/