Resolución de ecuaciones con SSM

Métodos numéricos avanzados

Grupo 2: Clozza, Nicolás - Della Sala, Rocío - Mamone, Federico - Rodríguez, Ariel - Santoflaminio, Alejandro

Instituto Tecnológico de Buenos Aires

Profesores: Álvarez, Adrián Omar - Fierens, Pablo Ignacio - Schmidberg, Pablo Esteban

RESUMEN

Objetivos. Realizar un programa en código Matlab para la resolución de la ecuación de Kuramoto–Sivashinsky mediante el uso de Spectral Splitting Methods (SSM).
Metódos. Se compararon el método de Lie Trotter, Strang y los métodos afines simétricos y asimétricos
Resultados. Existen diferentes variables como el orden o el paso que con distintas configuraciones nos permiten obtener mejores resultados.

Key words. SSM – KS – Kuramoto – Sivashinsky

1. Introducción

El presente trabajo consistió en la implementación y desarrollo de un programa en lenguaje Matlab para la resolución de la ecuación de Kuramoto-Sivashinsky (KS):

$$u_t = -uu_{xx} - u_{xx} - u_{xxxx} \quad x \in [0, 32\pi] \tag{1}$$

Para resolver la misma se hizo uso de métodos muy eficientes y precisos llamados Spectral Splitting Methods (SSM). Estos consisten en operar la ecuación a partir de las transformadas discretas de Fourier, separándola en dos subproblemas correspondientes a la parte lineal y a la no lineal.
Se implementaron los siguientes métodos para poder comparar sus resultados entre sí: Lie Trotter, Strang y métodos afínes (simétricos y asimétricos).

2. Implementación

El programa está desarrollado en Matlab, y puede ser ejecutado de la forma que se indica en el archivo *README.md*.
Para la implementación del sistema se hizo un uso parcial de los códigos provistos por la cátedra que se encuentran tanto en campus como en los diversos papers. Además se implementó una versión propia de la transformada rápida de Fourier (así como de su inversa), es decir que no se hizo uso de las funciones *fft* e *ifft* de Matlab.
El programa fue ejecutado y probado siempre en Matlab en una computadora a modo local.

3. Instrucciones de ejecución

Como se explicó anteriormente, para ser ejecutado es necesario leer la explicación que se detalla en el archivo *README.md*. La función principal, llamada *main*, se encuentra en el archivo *ks.m*. Este código se encuentra parcialmente basado en el paper provisto por la cátedra escrito por Kassam y Trefethen [1]. A modo explicativo, vale la pena destacar algunos de los parámetros utilizados en esta clase los cuales pueden ser interesantes modificar:

1. ***method***: Método utilizado.
2. ***h***: Paso utilizado.
3. ***pert***: Perturbación (para hacer pequeñas perturbaciones aleatorias).
4. ***q***: Orden (para aquellos métodos en cuales sea relevante cambiarlo).

4. Metodología

Como condición inicial se utilizó:

$$u(x, 0) = cos(x/16)(1 + sin(x/16)) \tag{2}$$

Se podía separar la misma en dos ecuaciones parciales, lineal y no lineal, las cuales nos habilitaban a encontrar soluciones aproximadas al problema mencionado. Para ello se llamó a cada expresión flujo ϕ_0 y flujo ϕ_1 , asociados a cada uno de los problemas parciales, siendo $\phi(h, u_0) = e^{h.A}u_0$. Es útil debido a que pudieron resolver fácilmente los problemas parciales, obteniendo los flujos asociados a cada problema parcial de la ecuación a resolver (que se detallarán luego) y aplicando composiciones de integradores de Lie Trotter.

4.1. Métodos implementados

A continuación se describirán los métodos que fueron implementados en el trabajo, los cuales están basados en los códigos mostrados en clase el día 18/10/2019.

4.1.1. Lie Trotter

Lie Trotter es un método de primer orden. Se encuentra implementado directamente en la clase principal ks.m. Está definido según la siguiente fórmula:

$$\Phi_{Lie}(h, u) = \phi_1(h, \phi_0(h, u)) \tag{3}$$

4.1.2. Strang

El método de Strang, de segundo orden, también implementado directamente en la clase principal ks.m. Está definido según la siguiente fórmula:

$$\Phi_{Strang}(h,u) = \phi_0(h/2, \phi_1(h/2,u)) \tag{4}$$

4.1.3. Afín Simétrico

El método afín asimétrico fue implementado en sus versiones en serie y en paralelo y pueden ser ejecutados con un orden arbitrario q, fijado en la clase ks.m. Está definido según la siguiente fórmula:

$$\Phi_{Asymmetric}(h) = \sum_{m=1}^{s} \gamma_m \phi_m^{\pm}(h/m) \tag{5}$$

Ambas implementaciones requirieron calcular los gammas. Afín Simétrico hace uso del método de Lie Trotter de forma iterada para ir calculando de forma sucesiva.

En el caso de la implementación en paralelo se hizo uso de la función *spmd* de Matlab para crear un pool paralelo.

4.1.4. Afín Asimétrico

Similar al método anterior, la principal diferencia tiene que ver con como se utilizó el método de Lie Trotter en los llamados sucesivos. Está definido según la siguiente fórmula:

$$\Phi_{Symmetric}(h) = \sum_{m=1}^{s} \gamma_m (\phi_m^{+}(h/m) + \phi_m^{-}(h/m)) \tag{6}$$

También requirió de un cálculo de gammas y para la versión en paralelo se utilizó la misma metodología de creación de un pool paralelo.

Para el caso de los métodos afines se definieron los integradores de la siguiente manera:

$$\Phi^{+}(h) = \phi_1(h)\ o\ \phi_0(h) \tag{7}$$

$$\Phi^{-}(h) = \phi_0(h)\ o\ \phi_1(h) \tag{8}$$

$$\Phi_m^{\pm}(h) = \phi^{\pm}(h)\ o\ \phi_{m-1}^{\pm}(h) \tag{9}$$

Además, en ambos casos, los gammas debieron cumplir una par de condiciones para asegurar que los integradores (6) y (7) son convergentes con orden q.

4.2. Resolución del problema

Si retomamos la ecuación (1), para trabajar con la misma vamos a transformarla aplicando Fourier, con lo cual nos queda:

$$\widehat{u_t} = -\frac{ik}{2}\widehat{u_2} + (k^2 - k^4)\widehat{u} \tag{10}$$

Esta ecuación se separó en dos partes: la parte lineal (segundo término de la suma de la ecuación (10)) y no lineal (primer término de la suma de la ecuación (10)).

La ecuación de la parte lineal pudo resolverse de forma analítica pero para resolver la ecuación correspondiente a la parte no lineal fue necesario utilizar la transformada rápida de Fourier (Fast Fourier Transform o FFT) y Runge-Kutta 4.

4.3. Transformada de Fourier

Para resolver la parte no lineal, se implementaron tanto la transformada de fourier como su inversa. Para ello se trabajó con los códigos provistos en [3].

5. Resultados

5.1. Tablas

A continuación se muestra en una tabla los resultados de diversas ejecucuciones del programa implementado. El cálculo de los errores locales que se muestran en las tablas se obtuvo a partir de restar a la aproximación obtenida otra conseguida con un orden mayor. Por otro lado el cálculo de los errores globales se obtuvo restando una aproximación obtenida con un paso con otra con un paso mayor. En todos los casos se utilizó tmax = 150.

5.1.1. Comparativa de exactitud obtenida entre diferentes métodos

La obtención del resultado se realizó calculando el error global con los pasos 0.002 y 0.004. En el caso de los métodos afines se trata de su implementación en serie.

Tabla 1

Método	Error Global	Error Local
Lie Trotter	4.56	75.28
Strang	3.35	107.46
Afín Simétrico (orden 4)	35.82	91.62
Afín Asimétrico (orden 4)	29.11	119.88

5.1.2. Comparativa entre distintos órdenes

La obtención del resultado se realizó calculando el error local para los métodos afines en serie con paso 0.25.

Tabla 2

Método	Orden	Error Local
Afín Simétrico	2	138.14
Afín Simétrico	4	91.62
Afín Asimétrico	2	148.36
Afín Asimétrico	4	137.19
Afín Asimétrico	6	96.95

En el cuadro anterior, se puede notar que un mayor orden nos permite encontrar resultados con un menor error. Sin embargo el orden no es la única variable a tener en cuenta para variar el error del resultado como se verá a continuación.

5.2. Comparativa entre distintos pasos

La obtención del resultado se realizó calculando el error local para los métodos afines en serie de orden 4.

Tabla 3

Método	Paso	Error Local
Afín Simétrico	0.25	91.62
Afín Simétrico	0.025	84.23
Afín Asimétrico	0.25	137.19
Afín Asimétrico	0.025	57.75

En la tabla anterior, se puede observar que en principio a menor paso, obtenemos resultados más precisos con menor error. Sin embargo, no se evaluaron pasos menores a 0.025 porque usar un paso muy pequeño requería de tiempos de procesamiento mayores.

5.2.1. Comparativa entre métodos afines en serie y paralelo

La obtención del resultado se realizó calculando el tiempo de cómputo para los métodos afines en serie y en paralelo con paso 0.025.

Tabla 4

Método	Orden	Tiempo de cómputo [s]
Afín Simétrico Serie	4	119
Afín Simétrico Paralelo	4	783
Afín Asimétrico Serie	4	220
Afín ASimétrico Paralelo	4	773

En el cuadro anterior notamos que obtenemos mejores tiempos de cómputo al correr los métodos afines en serie. En este caso se utilizó un paso 0.025. Se considera que los resultados cambiarían si se pudiera utilizar una mayor cantidad de trabajadores en el modo paralelo.
En este caso al utilizar Matlab de prueba en una computadora local, solo se nos permitía utilizar un máximo de 4 workers, uno por cada núcleo del procesador de la computadora. Es posible que los métodos en paralelo dieran mejores resultados de haber podido cambiar esta limitación.

5.2.2. Comparativa del Speed Up

La obtención del resultado se realizó calculando el speed up variando el orden para los métodos afines en paralelo con paso 0.025.

Tabla 5

Método	Orden	Speed Up
Afín Simétrico	2	0.08
Afín Simétrico	3	0.08
Afín Simétrico	4	0.17
Afín Asimétrico	2	0.18
Afín Asimétrico	3	0.23
Afín Asimétrico	4	0.23

Con respecto a la tabla anterior definimos Speed Up como $S = \frac{T_1}{T_n}$ siendo T_1 el tiempo de cómputo para el método en su versión en serie y T_n el tiempo de cómputo para el método en su versión en paralelo con n *workers*.
Observamos que a mayor orden, mayor speed up. Sin embargo, estos resultados, al igual que los anteriores entendemos que serían distintos si pudieramos probar mayores órdenes en los métodos paralelizables.
Lo que si podemos notar es que al incrementarse el speed up, en este caso significó que propocionalmente la diferencia entre los tiempos en serie y paralelo se achicaba.

5.3. Simulaciones

Con respecto a las perturbaciones, a modo de ejemplificación, se muestran a continuación dos figuras. La primera corresponde a la simulación obtenida con el método de Lie Trotter sin perturbación alguna. La segunda fue generada también con el mismo método, pero esta vez con perturbación.

5.3.1. Simulación sin perturbación

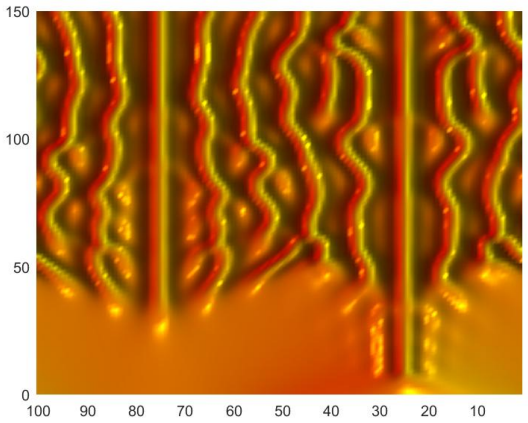

Figura 1

5.3.2. Simulación con perturbación

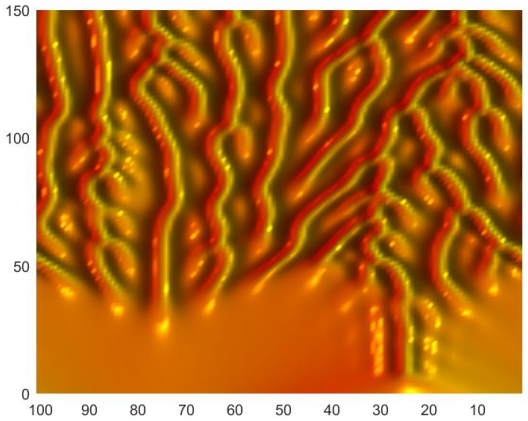

Figura 2

6. Conclusiones

1. Un mayor orden nos permitió bajar el error.
2. Un menor paso también permitió bajar el error.
3. No se llegó a una conclusión con los tiempos de cómputo entre los métodos afines en serie y paralelo debido a limitaciones propias. Puede que con comparando mayores órdenes y siendo posible la utilización de muchos núcleos, entonces las implementaciones en paralelo sean más rápidas que las versiones en serie.

Referencias

[1] Trefethen L.N. Kassam A.K. Fourt order time stepping for stiff PDE's., volume 26 No 4 pages: 1214-1233. SIAM J. SCI Comput.

[2] Alvarez A. Rial D. Affine combination of splitting type integrators implemented with parallel computing methods. International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering.

[3] Python code for FFT `https://rosettacode.org/wiki/Fast_Fourier_transform`