An introduction to the Split Step Fourier Method using MATLAB

Research · September 2015

DOI: 10.13140/RG.2.1.1394.0965

1 author:

Pablo Suarez
Delaware State University
13 PUBLICATIONS **90** CITATIONS

An introduction to the Split Step Fourier Method using MATLAB

Pablo U. Suárez

January 18, 2013

Abstract

The Split Step Fourier Method provides an excellent methodology for learning and teaching how to solve time dependent partial differential equations. This method is ubiquitously used in engineering and physics applications. In this article, we present the simplicity of this method by solving the Linear and Nonlinear Schrödinger Equation and with the aid of MATLAB, we demonstrate a straight forward implementation. Extension to other type of problems are also discussed with the corresponding codes. This article illustrates a very powerful numerical technique that can be implemented quite easily.

1 Introduction

The Split Step Fourier Method(SSFM) is a part of the the family of pseudo-spectral methods used to solve time dependent nonliner partial differential equation. The approach followed by SSFM follows the old age strategy of divide and conquer. The nonlinear PDE is divided into related sub-problems of the original equation. The method is easy to understand and with the use of MATLAB the implementation is straight forward.

In this article, we shall illustrate the implementation of the Fourier Split Method using MATLAB. We shall discuss the numerical solution of a variety of examples. We begin with the one dimesional linear Schrödinger Equation, the nonliner Schrd̈oinger equation, systems of the Schrodinger and the 2D Schrödinger equation. For a more detailed discussion on the origins and development of the method, the reader may wish to refer to books listed in

the references such as [7], [8] and [9]. We shall also assume that the reader has a basic knowledge of MATLAB programming, although the reader may also wish to consult a standard text on MATLAB such as [10].

2 Solving the Linear Schrödinger Equation by SSFM

To illustrate the method we consider the prototypical example of solving the Linear Schrödinger Equation (LSE). This equation appears in many applications such as quantum mechanics, electromagnetism, telecommunications and other fields. The non-dimensional time depent Linear Schrödinger equation is as follows:

$$i\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial^2 u}{\partial x^2} + V(x)u, \quad -\infty < x < \infty \qquad (1)$$
$$u(x,0) = u_0(x)$$

Here u is a complex valued function, $V(x)$ a real function and $i^2 = -1$. In solving this nonliner PDE one requires that the boundary condtions go to 0 as $x \to \pm\infty$ that is $\lim_{x\to\pm\infty} u(x,t) \to 0$.

It is known that for different values of the "potential" , $V(x)$, one cannot find a general solution. Instead one needs to resort to numerical methods for solving this PDE.

Splitting of the LSE It is convinient to rewrite equation (1) as :

$$\frac{\partial u}{\partial t} = i\mathcal{L}u + i\mathcal{N}u, \quad -\infty < x < \infty, \qquad (2)$$

where

$$\mathcal{L} = -\frac{1}{2}\frac{\partial^2}{\partial x^2} \text{ and } \mathcal{N} = -V(x). \qquad (3)$$

Consider a small interval of time, say $[\tau, \tau + \Delta t]$. Now assume that $u(\tau, x)$ is known and assume that $\mathcal{L}$ is negligible, that is $u_{xx} \approx 0$. Then (1) reduces to

$$i\frac{\partial u}{\partial t} = \mathcal{N}u \qquad (4)$$

This approximation leads to an ordinary differential equation (ODE) in time. The solution to this ODE is known and the solution at time $t = \tau + \Delta t$ is given by

$$u(x, \tau + \Delta t) = \exp\left(i\Delta t \mathcal{N}\right) u(x, \tau). \tag{5}$$

Next, we redo the problem but neglect the contribution of $\mathcal{N}$. For this case the PDE is linear with no variable coefficients,

$$\frac{\partial u}{\partial t} = i\mathcal{L}u. \tag{6}$$

This equation will be solved in the interval $[\tau, \tau + \Delta t]$ and use $u(x, \tau + \Delta t)$ as the initial condition. An analytical solution to this equation can be by taking the Fourier Transform

$$\frac{\partial \hat{u}}{\partial t} = i\mathcal{F}(\mathcal{L}u) = i\mathcal{F}\left(\frac{\partial^2 u}{\partial x^2}\right) = -ik^2\hat{u}. \tag{7}$$

In this work we use the following Fourier and Inverse Fourier Transform definitions

$$\hat{h}(k) = \mathcal{F}(h) = \int_{-\infty}^{\infty} h(x)e^{-2\pi ikx}dx \tag{8}$$

$$h(x) = \mathcal{F}(\hat{h}) = \int_{-\infty}^{\infty} \hat{h}(k)e^{2\pi ikx}dk \tag{9}$$

The partial derivatives are now gone and the equation becomes an ODE. The analytical solution can be computed and value of $\hat{u}(x, \tau + \Delta t)$ is given by

$$\hat{u}(k, \tau + \Delta t) = \exp\left(-ik^2\Delta t\right)\hat{u}(k, \tau) \tag{10}$$

The solution in the x domain can be calculated by inverse of the Fourier Transform:

$$u(x, \tau + \Delta t) = \mathcal{F}^{-1}(\exp(-ik^2\Delta t)\hat{u}(k, \tau + \Delta t)) \tag{11}$$

The entire procedure can be stated in a single equation:

$$u(x, \tau + \Delta t) = \mathcal{F}^{-1}\left(\exp\left(-ik^2\Delta t\right) \cdot \mathcal{F}\left(\exp\left(i\Delta t V(x)\right) u(x, \tau)\right)\right) \tag{12}$$

The essence of the split method is to think of advancing in time and applying $\mathcal{L}$ and $\mathcal{N}$ one at a time to complete each time step.

3 Implementation

In this section we discuss the algorithm of SSFM, the MATLAB implemetation of it and show some numerical experiments.

3.1 SSFM Algorithm and Implementation

We now discuss the implementation of the Split Step Fourier Method. u is solved by first multipliying it by $\exp(i\Delta t V(x))$. Then taking Fourier Transforms of u and finally taking the Inverse of the Fourier Transform.

In the language of a computer, $u^{(n)}$ is a vector that is being updated first by multipliying it by the expontential of a . The discrete analog of the Fourier Transform and its inverse are the so called Discrete Fourier Transforms (FDT) and Inverse Discrete Fourier Transform (IFDT). MATLAB can easily manipulate arrays operations and the DFTs and IFDTs are calculated by the bulitin functions Fast Fourier Transform (`fft`) and Inverse Fast Fourier Transform (`ifft`). Algorithm (3.1) suggest how this procedure will go.

Algorithm 3.1: SSFM(u_0, u^n, M)

$u^{(n)} \leftarrow u_0$
$k \leftarrow \frac{2\pi}{L}(-N/2 : 1 : N/2 - 1)$
for $n \leftarrow 1$ **to** M

$$\textbf{do} \begin{cases} u^{(n+1/2)} \leftarrow \exp(i\Delta t V(x))\, u^{(n)} \\ \hat{u}^{(n+1/2)} \leftarrow \mathcal{F}(u^{(n+1/2)}) \\ \hat{u}^{(n+1)} \leftarrow \exp(-i\Delta t k^2)\hat{u}^{(n+1/2)} \\ u^{(n+1)} \leftarrow \mathcal{F}^{-1}(\hat{u}^{(n+1)}) \\ u^{(n)} \leftarrow u^{(n+1)} \end{cases}$$

return (u^n)

In algorithm(3.1) one should be understand that $u^{(n)}$ is a vector and thus all the operations are component wise vector operations. This algorithm can be understood by begginers in scientific computing programming. A shorter version of Algorithm(3.2) can be obtained based on (12).

Algorithm 3.2: SSFM(u_0, u^n, M)

$u^{(n)} \leftarrow u_0$
for $j \leftarrow 1$ **to** M
  $u^{(n)} \leftarrow \mathcal{F}^{-1}\left(\exp\left(-ik^2\Delta t\right) \cdot \mathcal{F}\left(\exp\left(i\Delta t V(x)\right) u^{(n)}\right)\right)$
return (u^n)

The beauty of MATLAB can now be revelad when implementing either Algorithm (3.1) or (3.2). MATLAB's syntax permits almost "copy paste" procedure of the algorithms. The reader can see this in the respective implementation of these codes. In particular, Algorithm (3.2) like the pseudocode consists of only one line inside a **for**-loop!

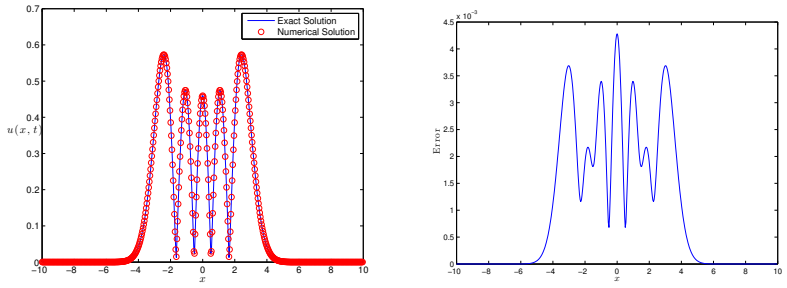
3.2 Numerical Results for 1D LSE

To convince the reader that SSFM is an accurate numerical scheme we present some numerical examples. We compare known analytical results of the Schrödinger equation with the results obtained by SSFM. It should be noted that there are rigorous arguments that concretely prove that this method is numerically stable and second order accurate. We do not carry this analysis as it requires more mathematical depth that might not be suitable for undergraduates, interested readers can see **citation needed** and references therein.

In this subsection we disscuss the performance of SSFM for 1 Dimensional Schrördinger Equation. We compare the numerical results with known analytical solutions and the calculated numerical solution.

4 Extension to other type of problems

In this section we discuss the extension of the SSFM to Nonlinear Problems, Coupled Nonlinear Systems and Higher dimensional problems.



(a) Exact Solution vs Analytical (b) Error of the Solution

Figure 1: Linear Schrödinger Solution and Error at $t = 2$

4.1 Nonlinear Problems

We illustrate the procedure by solving the Nonlinear Schrödinger Equation:

$$i\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial^2 u}{\partial x^2} - |u|^2 u, \tag{13}$$

$$u(x,0) = u_0(x) \tag{14}$$

This equation can be written as a general nonlinear evolution equation of form

$$i\frac{\partial u}{\partial t} = (\mathcal{L} + \mathcal{N})u, \tag{15}$$

$$u(x,0) = u_0(x) \tag{16}$$

where $\mathcal{L} = -\frac{1}{2}\frac{\partial^2}{\partial x^2}$ and $\mathcal{N} = -|u|^2$ are time independent linear and nonlinear operators, respectively. To solve this problem we follow the same procedure as before and split the problem into two parts. The "nonlinear" step is given by

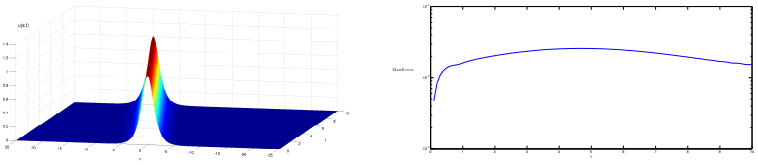
$$i\frac{\partial u}{\partial t} = \mathcal{N}u. \tag{17}$$

To advance the nonlinear part take $\mathcal{N} = q|u(x,\tau+\Delta t)|^2 \approx q|u(x,\tau)|^2$ and thus the analytical solution is given by

$$u(x,\tau+\Delta t) \approx \exp\left(i\Delta t\mathcal{N}\right)u(x,\tau) = \exp\left(iq\Delta t|u(x,\tau)|^2\right)u(x,\tau). \tag{18}$$

The linear step is given by

$$i\frac{\partial u}{\partial t} = \mathcal{L}u. \tag{19}$$



(a) Numerical Solution (b) Max error at each time step

Figure 2: Non-Linear Schrödinger Solution and Error

The linear part for this case has not changed and can be handled again by taking Fourier Transform. If $\mathcal{L}$ is a differential operator with constant coefficient only then we have

$$u(x, \tau + \Delta t) = \mathcal{F}^{-1}\left(\exp\left(i\Delta t\hat{\mathcal{L}}\right) \cdot \mathcal{F}\left(\exp\left(i\Delta t\mathcal{N}\right) u(x, \tau)\right)\right) \tag{20}$$

The code used in section 3 is modified and can be seen in the appendix.

4.2 Numerical Results for 1D NLSE

The numerical results show promise.

4.3 Higher Dimensional Problems

As a final example we show SSFM for 2D problems. We use the 2D version of the LSE to illustrate this extension. After reading this last example the reader can reason that this method can be extended to 2D with nonlinearities or higher order dimensional problems. If is also worth stating that one can use this method for systems of PDEs, (see for instance).

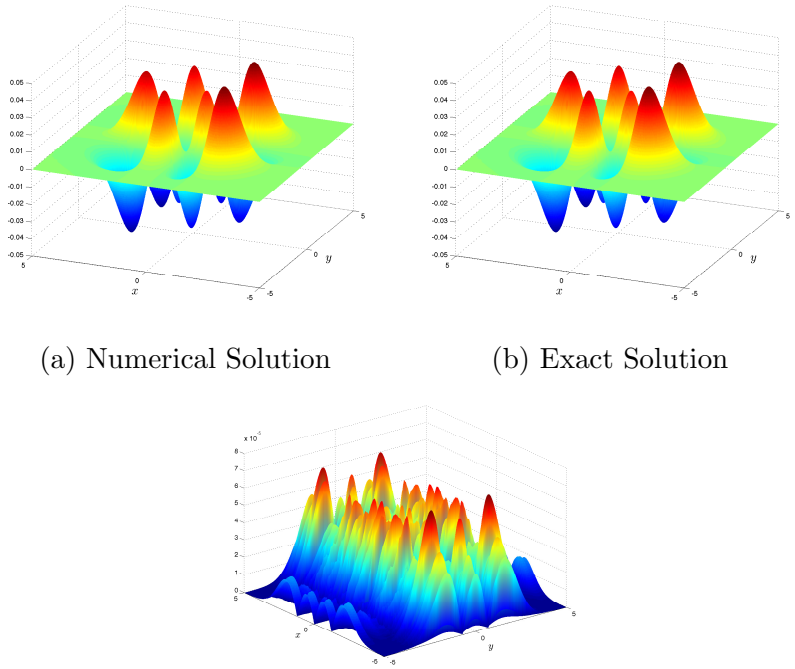
The 2D version of the LSE is given by:

$$i\frac{\partial u}{\partial t} = -\frac{1}{2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u + V(x, y)u \tag{21}$$

The SSFM is employed by letting

$$\mathcal{L} = -\frac{1}{2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) \tag{22}$$

$$\mathcal{N} = V(x, y) \tag{23}$$



(a) Numerical Solution

(b) Exact Solution

(c) Relative Error at time step $t = 6$

Figure 3: Non-Linear Schrödinger Solution and Error

5 Other Nonlinear Equations

The discussion here has been limited only to the nonlinear Schrodinger [2, 7] equation. However other time dependent nonlinear/linear problems can be solved by this method.

5.1 Benjamin Ono Equation

In this last example we use the Benjamin Ono equation to show how SSFM is applied. This example might seem exotic for undergraduate students.

The B-O equation is given by

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \mathcal{H}\left\{\frac{\partial^2 u}{\partial x^2}\right\} = 0, \quad -\infty < x < \infty, t > 0 \tag{24}$$

where $\mathcal{H}$ is the Hilbert Transform definde as

$$\mathcal{H}\{f\}(x) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{f(y)}{y-x}dy$$

together with initial and boundary conditions as

$$u(x,0) = f(x) \text{ and } u, \frac{\partial u}{\partial x} \to 0 \text{ as } |x| \to \infty.$$

6 Conclusions

In this article we presented the SSFM for solving linear and nonlinear Schrödinger equation. Algorithms and MATLAB implementation were presented for SSFM for both these equations. The algorithm showed the simplicity of the method for both cases and thanks to MATLAB the implementation was shown to be direct. Numerical examples were also shown to show the efficiency of the method. We hope that this article can serve as stepping stone for introductiong SSFM to undergraduates in a scientific computing course or as a self study tutorial for advanced undergraduates/beginning graduate conducting research.

References

[1] K-C Ang. Introducing the boundary element method with Matlab. *International Journal of Mathematical Education in Science and Technology*, 39(4):505–519, 2008.

[2] R Becerril, R S Guzmán, A Rendón-Romero, and S Valdez-Alvarado. Solving the time-dependent Schödinger equation using finite difference methods. *Revista Mexicana de Física*, 54(2):120–132, 2008.

[3] J Boyd. *Chebyshev and Fourier Spectral Methods*. Dover, New York, 2001.

[4] Y W Kwon and H Bang. *The Finite Element Method using Matlab*. CRC Press, New York, 1st edition, 1996.

[5] G M Muslu and H A Erbay. Higher-order split-step fourier schemes for the generalized nonlinear schrödinger equation. *Mathematics and Computers in simulation*, 67:581–595, 2005.

[6] T R Taha and M J Ablowitz. Analytical and numerical aspects of certain nonlinear evolution equations. II. Numerical, nonlinear Schrödinger equation. *Journal of Computational Physics*, 55(2):203–230, 1984.

[7] T R Taha and X Xu. Parallel Split-Step Fourier Methods for the coupled nonlinear Schrö dinger type equations. *The Journal of Supercomputing*, 5:5–23, 2005.

[8] JAC Weideman and B M Herbst. Split-step methods for the solution of the nonlinear Schrödinger equation. *SIAM Journal on Numerical Analysis*, 23:485–507, 1986.

[9] X-S Yang. *An Introduction to Computational Engineering with MATLAB*. Cambridge International Science Publishing, 1st edition, 2006.

Appendix

In this appendix we present the codes used in this paper. All of the codes required the standard MATLAB package and it has been tested to work with the student version as well.

LSE SSFM Implementation

```
N      = Q;                 % Number of Fourier modes
dt     = .005;              % Size of time step
tfinal = 2;                 % Final time
M      = round(tfinal/dt);  % Total number of time steps
J      = 100;               % Steps between output
L      = 20;                % Space period
q      = -1;
mu     = 2/3;
v      = 1;

h  = L/N;                   % Space step
n  = (-N/2:1:N/2-1)';       % Indices
x  = n*h;                   % Grid points

u0 = (12 - 48*x.^2 + 16*x.^4)./...
     (8.*sqrt(6).*exp(x.*x/2.)*pi.^0.25)
u  = u0;                    % Initial Condition
k  = 2*n.*pi/L;             % Wavenumbers.
t  = 0;

for m = 1:1:M               % Start time loop

u = exp(dt/2*1j*q*x.^2).*u; % Solve non-constant part of LSE
c = fftshift(fft(u));       % Take Fourier transform
c = exp(dt/2*1j*-k.^2).*c;  % Advance in Fourier Space
u = ifft(fftshift(c));      % Return to Physical Space

end
```

NLSE SSFM Implementation

```
N      = 512;               % Number of Fourier modes
dt     = .001;              % Time step
tfinal = 10;                % Final time
M      = round(tfinal./dt); % Total number of time steps
J      = 100;               % Steps between output
L      = 50;                % Space period
h      = L/N;               % Space step
n      =( -N/2:1:N/2-1)';   % Indices
x      = n*h;               % Grid points
u      = exp(1i*x).*sech(x);%Intial condition
k      = 2*n*pi/L;          % Wavenumbers.

for m = 1:1:M               % Start time loop

u  = exp(dt*1i*(abs(u).*abs(u))).*u;   % Solve non-linear part of NLSE
c  = fftshift(fft(u));                % Take Fourier transform
c = exp(-dt*1i*k.*k/2).*c;            % Advance in Fourier space
u = ifft(fftshift(c));                % Return to Physical Spacee

end
```

2D LSE SSFM Implementation

```
N       = 256;               % Number of Fourier modes
dt      = .0001;             % Time step
tfinal  = 6;                 % Final time
M       = round(tfinal/dt);  % Total number of time steps
L       = 10;                % Space period
h       = L/N;               % Space step
n       = [-N/2:1:N/2-1]';   % Indices
x       = n*h;               % Grid points along x
y       = n*h;               % Grid points along y
[X,Y]   = meshgrid(x,y);
u0      = (exp(-X.^2/2 - Y.^2/2).*(-2 + 4*X.^2).*...
          (-12*Y + 8*Y.^3))/(8*sqrt(6*pi));
u       = u0;                % Initial condition
k       = 2*n*pi/L;          % Wavenumbers.
[Ex,Ey] = meshgrid(e,e);     % Wavenumbers along both directions.

for m = 1:1:M                  % Start time loop

u = exp(dt*1i*-1/2*(X.^2+Y.^2)).*u;  % Solve non-constant part of 2D-LSE
c = fftshift(fft2(u));               % Take 2D-Fourier transform
c = exp(dt/2*1i*(-Ex.^2-Ey.^2)).*c;  % Advance in Fourier space
u = ifft2(fftshift(c));              % Return to physical space

end
```

B-O Equation SSFM Implementation

```
N       =  256;                     % Number of Fourier modes
dt      = .001;                     % Time step
tfinal  = 6;                        % Final time
M       = round(tfinal/dt);         % Total number of time steps
L       = 200;                      % Space period
h       = L/N;                      % Space step
n       = [-N/2:1:N/2-1]';          % Indices
x       = n*h;                      % Grid points
kk      = 1/(2*sqrt(13));
cc      = .2;
u       = 4*cc./(1 + cc^2*x.^2);    % Compute IC
k       = 2*n*pi/L;                 % Wavenumbers.

for m = 1:1:M                       % Start time loop

u = exp(ifft(fftshift(-dt*1i*k.*... % Solve nonlinear part of B-O Eq.
         fftshift(fft(u))))).*u;
c = fftshift(fft(u));               % Take Fourier transform
c = exp(dt*1i*sign(n).*k.^2).*c;    % Advance in Fourier space
u = ifft(fftshift(c));              % Return to physical space

end
```