Fortran codes to compute $u(t) = \mathrm{e}^{-itH} u_0$ using an algorithm based on symplectic splitting methods*

Sergio Blanes[1†] Fernando Casas[2‡] Ander Murua[3§]

February 17, 2015

1 Introduction

The algorithm is mainly addressed to numerically solve the discretized Schrödinger equation, which is a linear system of ordinary differential equations (ODEs)

$$i\frac{d}{dt}u(t) = Hu(t), \qquad u(0) = u_0 \in \mathbb{C}^N, \tag{1}$$

where $u(t)$ represents a discretized version of the wave function at the space grid points and H is the $N \times N$ Hermitian Hamiltonian matrix. The spatial discretization restricts the energy range of the approximation and imposes an upper bound to the high frequency components represented by the discrete solution. The exact solution is of course

$$u(t) = \mathrm{e}^{-itH}\, u_0, \tag{2}$$

and $u(t)$ is approximated by methods which use polynomials (and thus only vector-matrix products). In this sense, the proposed methods are useful to numerically solve (2) even if this problem is not originated from the Schrödinger equation.

The following inputs have to be provided by the user:

- t: Time of integration.
- u_0: Initial conditions.
- $E_{\max}, E_{\min}$: Upper and lower bounds to the extreme eigenvalues of H.
- N: the dimension of the problem.
- $ProdAv(v, N, w)$: A subroutine, `ProdAv.f`, to be called by the algorithm to obtain w from v such that $w = Hv$ with $v, w \in \mathbb{R}^N$.

*This document is to be read jointly with Ref. [1]. The papers [2, 3] can also be useful to better understand how the new symplectic splitting methods are built. These notes are addressed to show how to use the algorithm.

†Email: `serblaza@imm.upv.es`
‡Email: `Fernando.Casas@uji.es`
§Email: `Ander.Murua@ehu.es`

- *tol*: The desired tolerance, in the sense that the approximate solution, $\tilde{u}$, must satisfy $\|\tilde{u} - u(t)\| < tol$.

We write

$$H = \alpha I + \beta \tilde{H}, \qquad \text{where} \qquad \alpha = \frac{E_{\max} + E_{\min}}{2}, \quad \beta = \frac{E_{\max} - E_{\min}}{2} \tag{3}$$

and $-1 \leq \sigma(\tilde{H}) \leq 1$. If we consider the integration for a time step of length $\tau \equiv \Delta t$ we have

$$u(\tau) = \mathrm{e}^{-i\tau H}\, u_0 = \mathrm{e}^{-i\tau\alpha}\, \mathrm{e}^{-i\tau\beta\tilde{H}}\, u_0. \tag{4}$$

Then we compute the vector-matrix products as follows:

$$\tilde{H}\, u_0 = \frac{1}{\beta} H\, u_0 - \frac{\alpha}{\beta} u_0$$

The methods compute approximations to the scaled and shifted problem

$$w(\tau) = \mathrm{e}^{-i\tau\beta\tilde{H}} u_0. \tag{5}$$

and finally the shift is applied.

1.1 Symplectic splitting pseudo-method: SplittingEx.f

We have the program `MainSplittingMethod.f` as an example for illustrating the use of the method. Next we reproduce the part of the code where the relevant data are introduced. These correspond to Example 1 in [1]:

$$N = 10000, \quad E_{\min} = 0, \quad E_{\max} = 2, \quad t = 20, \quad tol = 10^{-8}.$$

The initial conditions are stored in the file `InitialConditions.dat`.

```
               MainSplittingMethod.f
c N is the dimension of the problem
      PARAMETER(N=10000)
cccc Data to be provided by the user  ccccccccccccccccccccccccc
c Integration Time
      t=20.d0
c Bounds to the spectral radius
      Emin=0.d0;   Emax=2.d0
c Tolerance
      tol=10.d0**(-8.d0)
c Initial conditions
      OPEN (30,FILE='InitialConditions.dat',STATUS='unknown')
      do jj=1,N
            read (30,*) q0(jj),p0(jj)
      enddo
c Computing the solution using Splitting Methods
```

```
      call SplittingEx(t,Emin,Emax,tol,N,q0,p0,q,p,icost)

C Subroutines used
      include 'SplittingEx.f'
      include 'ProdAv.f'
```

The input data are: '`t,Emin,Emax,tol,N`', and the initial conditions '`q0,p0`'.

The subroutine `SplittingEx` returns the solution, '`q,p`', and the cost '`icost`' as the number of vector-matrix multiplications. This program uses the subroutine '`ProdAv.f`' provided by the user with the following arguments:

```
    call ProdAv(v,N,w)
```

that computes the vector-matrix product, `w=Hv`.

The relevant part of `SplittingEx.f` is:

```
c Number of coefficients for each method
      nmet(1)=10; nmet(2)=10; ...  nmet(19)=60;    nmet(20)=60;

      beta=(Emax-Emin)/2.d0;    rhoH=beta;
      alpha=(Emax+Emin)/2.d0;  shift=alpha
      theta=t*beta

      call ChoiceMethod(theta,tol,n1,imethod1,h1,n2,imethod2,h2)
      call CoefsSplittingEx(imethod1,a1,b1,imethod2,a2,b2)

      q = q0;    p = p0
      ha1=h1*a1;   hb1=h1*b1
      do i=1,n1
        call Split(q,p,N,imethod1,nmet(imethod1),ha1,hb1,beta,shift)
      enddo
      ha2=h2*a2;   hb2=h2*b2
      do i=1,n2
        call Split(q,p,N,imethod2,nmet(imethod2),ha2,hb2,beta,shift)
      enddo

      qs =  cos(shift*t)*q + sin(shift*t)*p
      ps = -sin(shift*t)*q + cos(shift*t)*p
      q=qs;   p=ps
```

We have a list of 20 methods where `nmet(i)` for `i=1,...,20`, stores the number of stages of each method. Next, α, β and θ are computed. From this information, the subroutine `ChoiceMethod` chooses the optimal methods labeled by `imethod1,imethod2`, to be used `n1,n2`, times with time steps `h1,h2`, respectively. Once the methods are chosen, the subroutine `CoefsSplittingEx`, load the coefficients of the corresponding methods into the vectors `a1,b1,a2,b2`.

Next, each step for each method is computed . The implementation applies the FSAL (first-same-as-last) property to save one vector-matrix product as follows:

```
q = q0;    p = p0
ha1=h1*a1;   hb1=h1*b1
call ProdAv(p,N,p1)
do i=1,n1
  q=q + ha1(1)*(p1/beta -  shift/beta*p )
  do j=1,nmet(imethod1)
      call ProdAv(q,N,q1)
      p=p - hb1(j)*(q1/beta -  shift/beta*q )
      call ProdAv(p,N,p1)
      q=q + ha1(j+1)*(p1/beta -  shift/beta*p )
   enddo
enddo
ha2=h2*a2;   hb2=h2*b2
do i=1,n2
  q=q + ha2(1)*(p1/beta -  shift/beta*p )
  do j=1,nmet(imethod2)
      call ProdAv(q,N,q1)
      p=p - hb2(j)*(q1/beta -  shift/beta*q )
      call ProdAv(p,N,p1)
      q=q + ha2(j+1)*(p1/beta -  shift/beta*p )
   enddo
enddo
```

which requires `2(n1*nmet(imethod1)+n2*nmet(imethod2))+1` real vector-matrix products (or `n1*nmet(imethod1)+n2*nmet(imethod2)+1/2` products). Finally, the shift is applied to the auxiliary vectors `qs,ps`, which lead to the approximate solution `q,p`.

The set of methods are collected in the directory `Coefficients`, which must be located in the same directory as the main program (or one has to indicate were it is allocated). The coefficients of each method are collected in separate files with suggested names:

- `coefs10Theta05.dat`: The 10-stage method with $\theta' = 0.5$, $M_{10}^{(0.5)}$.
- `coefs10Theta09.dat`: The 10-stage method with $\theta' = 0.9$, $M_{10}^{(0.9)}$.
- Etc.

NOTE: If new methods with different number of stages and tailored for different tolerances were obtained in the future, this algorithm will be adjusted properly for an optimal performance.

References

[1] S. Blanes, F. Casas, and A. Murua. An efficient algorithm based on splitting for the time integration of the Schrödinger equation. 2015. Submitted. Axiv.....

[2] S. Blanes, F. Casas, and A. Murua. On the linear stability of splitting methods. *Found. Comp. Math.*, 8:357–393, 2008.

[3] S. Blanes, F. Casas, and A. Murua. Error analysis of splitting methods for the time dependent Schrödinger equation. *SIAM J. Sci. Comput.*, 33:1525–1548, 2011.