Implementing Time-Splitting Spectral Methods

Harald Hofstätter

mailto:hofi@harald-hofstaetter.at

Outline

- **Basics** (spectral methods)
- **Implementation** (object oriented design)
- **Time-splitting methods** (higher order, adaptive)
- **Numerical example** (from project proposal)

Basics

Problem class

Consider evolution equation of the form

$$\frac{d}{dt}u(t) = A(u(t)) + B(u(t))$$
$$u(0) = u_0$$

where there exist efficient methods for the solution of the subproblems

$$\frac{d}{dt}u(t) = A(u(t)), \quad u(0) = u_0$$
$$\frac{d}{dt}u(t) = B(u(t)), \quad u(0) = u_0$$

Prototypical example: **Schrödinger equation**

$$\mathrm{i}\frac{d}{dt}u(x,t) = -\frac{1}{2}\Delta u(x,t) + V(x)u(x,t)$$

Subproblem $\mathrm{i}\frac{d}{dt}u(x,t) = -\frac{1}{2}\Delta u(x,t)$ to be solved by "spectral method"
Subproblem $\mathrm{i}\frac{d}{dt}u(x,t) = V(x)u(x,t)$ to be solved analytically (i.e., exactly)

Proper combination of the methods for the subproblems yields method for the full problem $\rightarrow$ **Time splitting spectral method**

Basics

Spectral method

Spectral method for the subproblem

$$\frac{d}{dt}u(t) = Au(t), \quad u(0) = u_0$$

A . . . linear self- (or skew-) adjoint operator on some given Hilbert space

There exists an orthonormal basis $\{\varphi_1, \varphi_2, \dots\}$ consisting of eigenfunctions of A

$$A\varphi_k = \lambda_k \varphi_k, \quad k = 1, 2, \dots$$

If expansion of initial data u_0 is given

$$u_0 = \sum_{k=1}^{\infty} c_k \varphi_k$$

then the solution of $\frac{d}{dt}u(t) = Au(t), \quad u(0) = u_0$ is

$$u(t) = \sum_{k=1}^{\infty} e^{t\lambda_k} c_k \varphi_j$$

Basics

Example

Example: Schrödinger equation

$$\mathrm{i}\frac{d}{dt}u(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}u(x,t) + V(x)u(x,t)$$

on 1D-domain $[0, 2\pi]$ with periodic boundary conditions

- Spectral method for subproblem $\mathrm{i}\frac{d}{dt}u(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}u(x,t)$ (free Schrödinger equation)

  Eigenfunctions of $A = \frac{1}{2}\mathrm{i}\frac{\partial^2}{\partial x^2}$ with periodic boundary conditions

  $$\varphi_k(x) = e^{\mathrm{i}kx}, \quad k = 0, \pm 1, \pm 2 \ldots \quad \text{(plane waves)}$$

  with eigenvalues

  $$\lambda_k = -\frac{\mathrm{i}}{2}k^2, \quad k = 0, \pm 1, \pm 2 \ldots$$

- Analytical solution for subproblem $\mathrm{i}\frac{d}{dt}u(x,t) = V(x)u(x,t)$, $u(0) = u_0$

  $$u(x,t) = e^{-\mathrm{i}tV(x)}u_0(x)$$

Full discretization:

- For an actual implementation on a computer the solution is computed on a finite set $\{x_0, \ldots, x_{N-1}\}$ of sample points only
  $\rightarrow$ "grid function" $\tilde{u} = (\tilde{u}_0, \ldots, \tilde{u}_{N-1})$, where $\tilde{u}_k = u(x = x_k)$
- Only finite expansions

$$u(x) = \sum_{k=1}^{N} \hat{u}_k \varphi_k(x)$$

  are considered
- Discrete Fourier transform: Find finite expansion which coincides ("collocates") with given grid function at sample points
- Inverse discrete Fourier transform: Evaluation of finite expansion at sample points

For Schrödinger equation on $[0, 2\pi]$ with periodic boundary conditions:

$$x_k = \frac{k}{2\pi}, \quad k = 0, \ldots, N-1, \qquad u(x) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{u}_k e^{ikx}$$

(Inverse) discrete Fourier transform implemented by FFT

Gross-Pitaevskii equation

$$\mathrm{i}\tfrac{d}{dt}u(x,t) = -\frac{1}{2}\Delta u(x,t) + V(x)u(x,t) + \kappa|u(x,t)|^2 u(x,t)$$

Subproblem

$$\mathrm{i}\tfrac{d}{dt}u(xt) = V(x)u(x,t) + \kappa|u(x,t)|^2 u(x,t),\ \ u(x,0) = u_0(x)$$

can be solved analytically

$$u(x,t) = e^{-\mathrm{i}t(V(x)+\kappa|u_0(x)|^2)}u_0(x)$$

Gross-Pitaevskii equation in $\mathbb{R}$ with additional harmonic potential

$$\mathrm{i}\tfrac{d}{dt}u(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}u(x,t) + \frac{\omega^2 x^2}{2}u(x,t) + V(x)u(x,t) + \kappa|u(x,t)|^2 u(x,t)$$

Subproblem $\mathrm{i}\tfrac{d}{dt}u(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}u(x,t) + \frac{\omega^2 x^2}{2}u(x,t),\ u(x,0) = u_0(x)$ (harmonic oscillator) can be solved by spectral method using expansion of u_0 into Hermite functions

Basics

Further examples

Gross-Pitaevskii equation in $\mathbb{R}^2$ with additional rotation term

$$\begin{aligned} \mathrm{i}\tfrac{d}{dt}u(x,y,t) = \\ -\frac{1}{2}\Delta u(x,y,t) + \frac{\omega^2}{2}(x^2+y^2)u(x,t) + \mathrm{i}\Omega(x\partial_x - y\partial_x)u(x,y,t) \\ +V(x)u(x,t) + \kappa|u(x,t)|^2 u(x,t) \end{aligned}$$

Using cylindrical coordinates the "green" suproblem can be solved by a spectral method with eigenfunctions involving generalized-Laguerre functions.

Movie: $V = 0.4y^2$, $\Omega = 0.5$, $\kappa = 100$, $\omega = 0.8$

$$u_0(x,y) = \frac{x+\mathrm{i}y}{\sqrt{\pi}} e^{-\frac{x^2+y^2}{2}}$$

100 Laguerre, 128 Fourier, Strang splitting ($\Delta t = 0.02$), $t_{\mathrm{end}} = 15$

Reference: H. Hofstätter, O. Koch, M. Thalhammer, *Convergence analysis of high-order time-splitting pseudo-spectral methods for rotational Gross-Pitaevskii equations*, Numer. Math. 127(2014), pp. 315-364.

Implementation

Object-oriented design, abstract base class for spectral methods

Implementaion based on an **object-oriented design**
Supported by modern versions of Matlab, Fortran, etc.
We give some simplified excerpts from our implementation in Matlab

Base class `spectral_method` is abstract
Concrete spectral methods implemented as subclasses of the base class

Base class specifies the methods which each concrete subclass has to implement; this technique allows implementation of *generic algorithms* which can be used for any concrete subclass

```
1 classdef spectral_method < handle
2     methods(Abstract)
3         init_wave_function(obj, U);
4         to_real_space(obj, wf);
5         to_fourier_space(obj, wf);
6         propagate_A(obj, wf, dt);
7         propagate_B(obj, wf, dt);
8     end
9 end
```

`init_wave_function` expects a function handle `U` and returns the values of this function at the sample points

Each of the other methods expects and returns an object of type `wave_function` (see next slide)

Implementation

`wave_function` class

An object of class `wave_function`

- has a (pointer to the concrete subclass of) `spectral_type`
- contains data
- and information whether this data is given in real or fourier space
- For each method of `spectral_method` (except `init_wave_functions`) it contains a corresponding "call back" method which calls the corresponding concrete method of the subclass with the current `wave_function` object as argument

Implementation

wave_function class (continued)

```
1  classdef wave_function
2      properties(Constant)
3          fourier_space = 0;
4          real_space = 1;
5      end
6
7      properties
8          U; % data
9          real_space_or_fourier_space;
10         spectral_type;
11     end
12
13   methods
14     %constructor
15     function obj = wave_function(U, spectral_method)
16       obj.U = spectral_method.init_wave_function(U);
17       obj.real_space_or_fourier_space = real_space;
18       obj.spectral_method = spectral_method;
19     end
20
21     function wf=to_fourier_space(wf) % call back
22       wf = wf.spectral_method.to_fourier_space(wf);
23     end
24     ..................
25   end
```

Implementation

Concrete realization of a spectral method

```
classdef gpe_fourier_1D < spectral_method
  properties
    N;        % length of Fourier expansions
    lambda;   % eigenvalues
    V;        % potential evaluated at sample points
    kappa;    % coupling
  end

  methods
    %constructor
    function o=gpe_fourier_1D(N, V, kappa)
      o.N = N;
      % function_handle V expected:
      o.V = V(2*pi/M*(0:M-1)');
      o.kappa = kappa;
      k = fftshift(-N/2:N/2-1)';
      o.lambda = -.5i*k.^2;
    end

    function U=init_wave_function(o, U)
      % function_handle U expected:
      U = U(2*pi/o.N*(0:o.N-1)');
    end
```

Implementation

Concrete realization of a spectral method (continued)

```
1     function wf=to_fourier_space(o, wf)
2       if wf.real_space_or_fourier_space==wf.real_space
3         wf.U = fft(wf.U)*(sqrt(2*pi/o.N);
4       end
5       wf.real_space_or_fourier_space = wf.fourier_space;
6     end
7
8     function wf=to_real_space(o, wf)
9       if wf.real_space_or_fourier_space==wf.fourier_space
10        wf.U = ifft(wf.U)*(o.N/sqrt(2*pi));
11      end
12      wf.real_space_or_fourier_space = wf.real_space;
13    end
14
15    function wf=propagate_A(o, wf, dt)
16      wf = o.to_fourier_space(wf);
17      wf.U = exp(dt*o.lambda).*wf.U;
18    end
19
20    function wf=propagate_B(o, wf, dt)
21      wf = o.to_real_space(wf);
22      wf.U = exp((-dt*1i)*(o.V+o.kappa*abs(wf.U).^2)).*wf.U;
23    end
24  end
25 end
```

Time-splitting methods

Strang splitting method

Time splitting spectral method for

$$\frac{d}{dt}u(t) = A(u(t)) + B(u(t)),\ u(0) = u_0$$

Combine (spectral) methods for the two subproblems
Idea: On a grid $\{t_0 = 0, t_1 = \Delta t, t_2 = 2\Delta t, \dots\}$ with stepsize Δt compute approximations $\{u_0, u_1 \approx u(t_1), u_2 \approx u(t_2), \dots\}$ to the solution $u(t)$ iteratively; given u_j compute u_{j+1} by the scheme

$$\text{solve } \frac{d}{dt}v(t) = A(v(t)),\ v(0) = u_j, \quad \text{set } u_j^* = v(\frac{1}{2}\Delta t)$$

$$\text{solve } \frac{d}{dt}v(t) = B(v(t)),\ v(0) = u_j^*, \quad \text{set } u_j^{**} = v(t)$$

$$\text{solve } \frac{d}{dt}v(t) = A(v(t)),\ v(0) = u_j^{**}, \quad \text{set } u_{j+1} = v(\frac{1}{2}\Delta t)$$

$\rightarrow$ second order **Strang splitting**, characterized by list of coefficients

$$\frac{1}{2}, 1, \frac{1}{2}$$

Can be easily implemented within our object oriented framework as a *generic* algorithm (i.e., implementation can be used for *any* concrete spectral method)

Time-splitting methods

Strang splitting test implemetation

```
1  N=256;
2  kappa=100; % cubic coupling parameter
3
4  %function handler for harmonic potential:
5  V = @(x) .5*(x-pi).^2;
6
7  my_spectral_method = gpe_fourier_1D(N,V,kappa);
8
9  %function handle for initial data:
10 U0 = @(x,t) exp(-.5*(x-pi).^2)/pi^.25;
11
12 dt = .001; % stepsize
13
14 u = wave_function(U0, my_spectral_method);
15 %Here u contains the initial data
16
17 u = u.propagate_A(0.5*dt);
18 u = u.propagate_B(dt);
19 u = u.propagate_A(0.5*dt);
20 %Here u contains the solution after one step
21 %of the Strang splitting scheme
22
23 %ensure data is given in real space
24 u = u.to_real_space();
```

Time-splitting methods

Higher order schemes

Lie-Trotter (1st order):

$$1, 1$$

Strang (2nd order):

$$\frac{1}{2}, 1, \frac{1}{2}$$

3rd order scheme:

$$\frac{7}{24}, \frac{2}{3}, \frac{3}{4}, -\frac{2}{3}, -\frac{1}{24}, 1$$

Yoshida (4th order):

$$\frac{1}{2(2-2^{1/3})}, \frac{1}{2-2^{1/3}}, \frac{1-2^{1/3}}{2(2-2^{1/3})}, \frac{-2^{1/3}}{2-2^{1/3}}, \frac{1-2^{1/3}}{2(2-2^{1/3})}, \frac{1}{2-2^{1/3}}, \frac{1}{2(2-2^{1/3})}$$

6th order scheme:

a_i	b_i
0.3922568052387786 32	0.7845136104775572 64
0.5100434119184576 99	0.2355732133593581 34
-0.4710533854097564 37	-1.1776799841788710 07
0.0687531682525201 06	1.3151863206839112 19
0.0687531682525201 06	-1.1776799841788710 07
-0.4710533854097564 37	0.2355732133593581 34
0.5100434119184576 99	0.7845136104775572 64
0.3922568052387786 32	

Time-splitting methods

Higher order schemes

Note: Schemes of order 3 or higher necessarily contain negative coefficients

- Suitable for evolution equation of (time-symmetric) *Schrödinger* type
- *Not* suitable for evolution equation of *parabolic* type

  Such equations have to be solved e.g. for the *imaginary time propagation method* for the computation of ground states ($t \to \mathrm{i}t$)

  Partial remedy: Schemes with complex coefficients with positive real part

  3rd order scheme:

a_i	b_i
$0.25 - 0.1443375672974064 41\mathrm{i}$	$0.5 - 0.2886751345948128 82\mathrm{i}$
0.5	$0.5 + 0.2886751345948128 82\mathrm{i}$
$0.25 + 0.1443375672974064 41\mathrm{i}$	

Time-splitting methods

Adaptive higher order schemes: step size control

Standard stepsize control from Hairer/Norsett/Wanner:
Given a tolerance tol, parameters

$$fac = 0.8, \quad facmin = 0.25, \quad facmax = 4.0$$

and an estimate for the current local error compute

$$err = \|\text{error estimate}\|/tol$$

$$\Delta t_{\text{new}} = \Delta t * \min(facmax, \max(facmin, fac \cdot (1/err)^{1/(p+1)}))$$

where p . . . order of method

If $err \leq 1$ **then** step size is accepted and solution advanced with Δt_{new} as step size

else steps size is rejected and above computation is repeated with $\Delta t = \Delta t_{\text{new}}$

Time-splitting methods

Adaptive higher order schemes: error estimators

- **Embedded pairs of splitting schemes**

  $$.5, 1.26376261582, .63188130791, 1, -.13188130791, -1.26376261582$$

  $$.5, 1, .5$$

  Example: 3(2) pair, higher order pairs exist

- **Defect based error estimators**

  Can be effectively computed for *any* given higher order splitting scheme

Numerical example

Example for the proposal of the WWTF project
"Adaptive Time-Splitting for Many-Body Quantum Propagation"

1D Gross-Pitaevskii equation

$$\mathrm{i}\tfrac{d}{dt}u(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}u(x,t) + V(x)u(x,t) + \kappa|u(x,t)|^2 u(x,t)$$

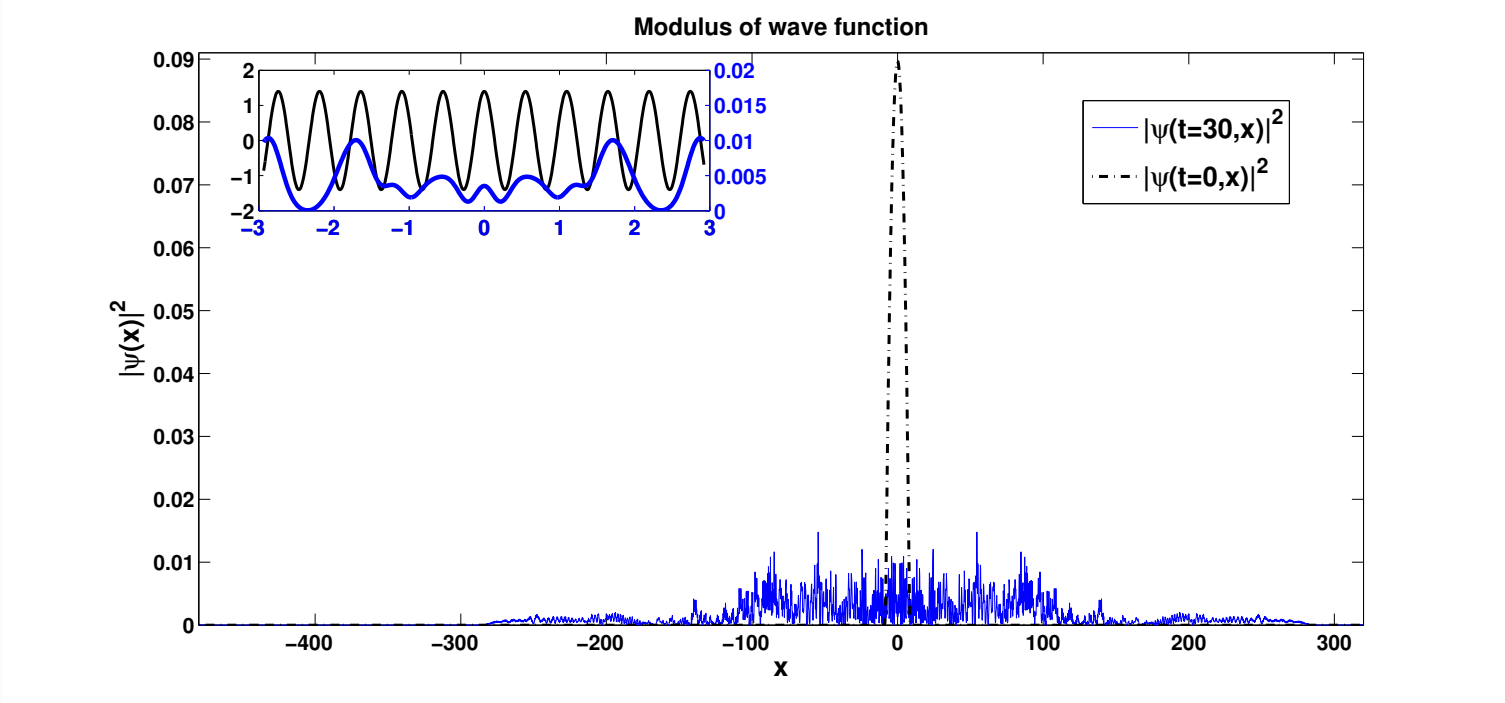
$$V(x) = 1.4 \cdot \cos(11.46x), \quad \kappa = 390$$

Initial state $u_0(x)$ ...ground state of a Bose-Einstein condensate described by this Gross-Pitaevskii equation with $V(x)$ replaced by the harmonic trap $\tilde{V}(x) = \frac{1}{2}x^2$
Effectively, by imaginary time propagation $(t \to \mathrm{i}t)$

$$u_0(x) = \lim_{t\to\infty} \frac{v(x,t)}{\|v(\cdot,t)\|_{L_2}}$$

$$\tfrac{d}{dt}v(x,t) = \frac{1}{2}\frac{\partial^2}{\partial x^2}v(x,t) - \frac{1}{2}x^2 v(x,t) - \kappa\frac{|v(x,t)|^2}{\|v(\cdot,t)\|_{L_2}^2}v(x,t)$$

Numerical example

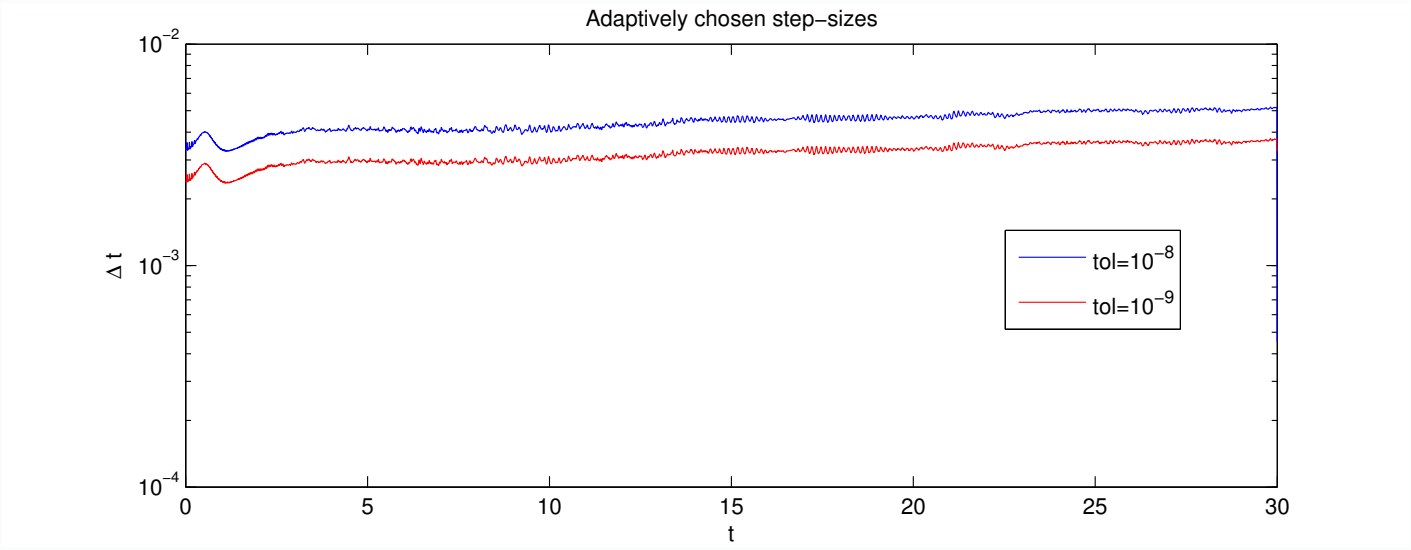


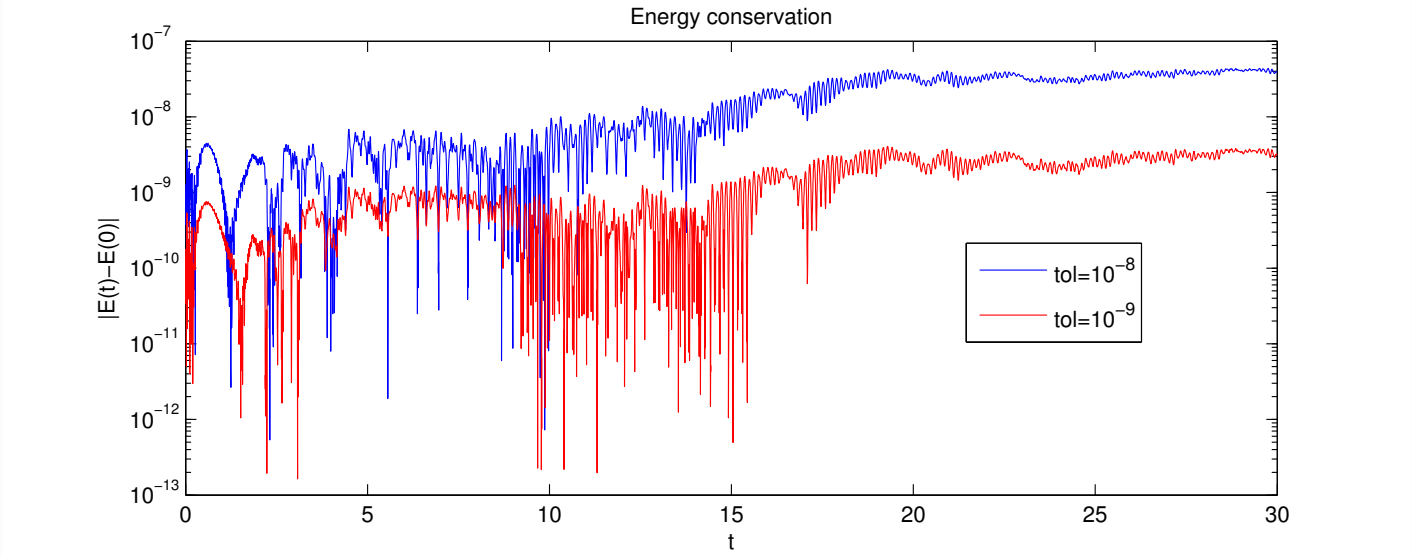
Parameters for numerical solution:

- 8192 basis functions on interval $[-400, 400]$
- 6th order time splitting scheme (16 FFTs/step)
- defect based error estimator (40 FFTs/step)

For $tol = 10^{-8}$ integration up to $t = 30$ requires ≈ 6800 steps

Numerical example





Thank you for your attention!

Implementing Time-Splitting Spectral Methods

Harald Hofstätter

mailto:hofi@harald-hofstaetter.at