In [1]:

```
using OrdinaryDiffEq
using FFTW
using LinearAlgebra
using Plots

include("../../code/TaylorFourier.jl")
include("NLS_ODE_DP5_fcns.jl")
include("NLS_ODE_TF_fcns.jl")
```

Out[1]:

```
NLS_ODE_TF_cache_init (generic function with 1 method)
```

Numerical solution of cubic NLS with Taylor-Fourier series

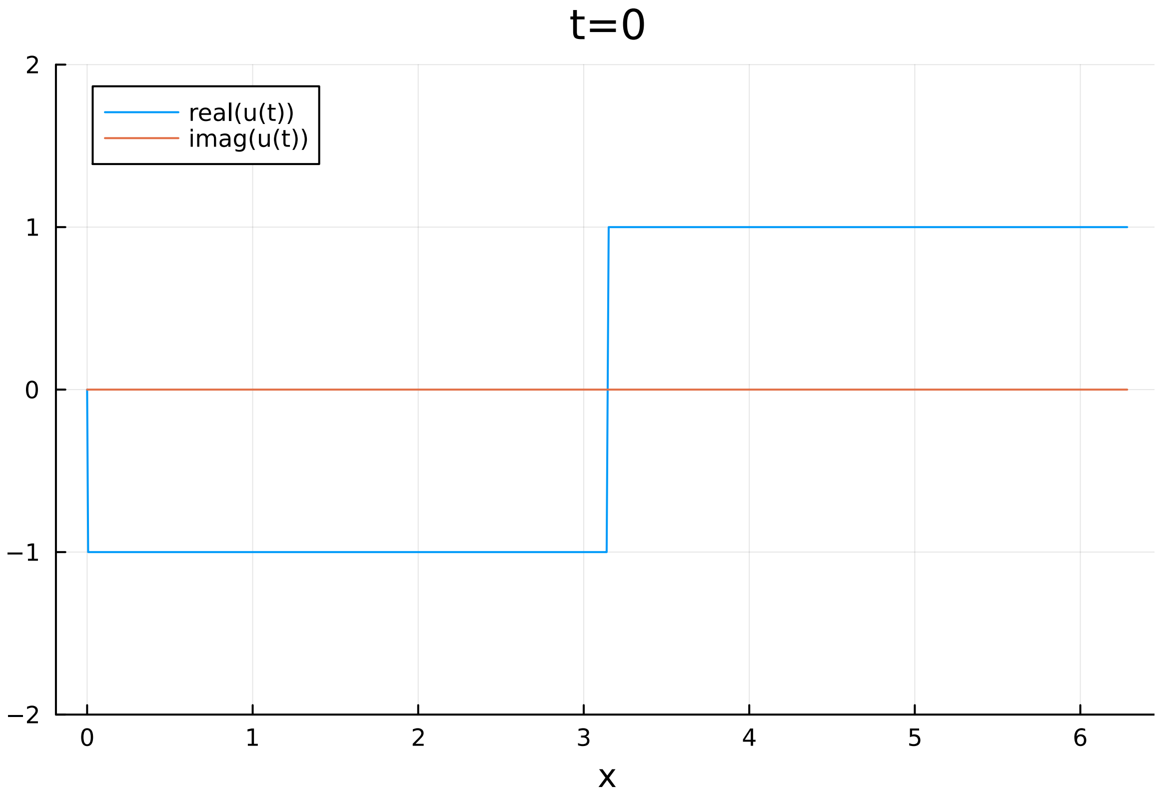
Numerical solution with a Lawson method based on a 5th order explicit RK method (due to Dormand and Prince)

We next solve numerically our initial value problem, with a given spatial semi-discretization, and show in the same figure the real part and the imaginary part of $u(t_{end})$.

In [2]:

```
J = 2^9
J2 = 2*J

epsilon = 1. # Try also epsilon=1e-1 and epsilon=1e-2


t_end = (1/epsilon - 1 + 0.1)*pi

tspan = (0.,t_end)
omega = 1.

W0=Vector{ComplexF64}(undef,J2)
W0[1]= zero(ComplexF64)
for i in 2:J
    W0[i]=-epsilon+0.0im
end
W0[J+1]= zero(ComplexF64)
for i in J+2:J2
    W0[i]=epsilon+0.0im
end



yrange=(-2epsilon,2epsilon)
xx = range(0,stop=2pi,length=J2)

plot(xlabel="x", title="t=0", ylims=yrange)
plot!(xx,real(W0),label="real(u(t))")
plot!(xx,imag(W0),label="imag(u(t))")
```

Out[2]:



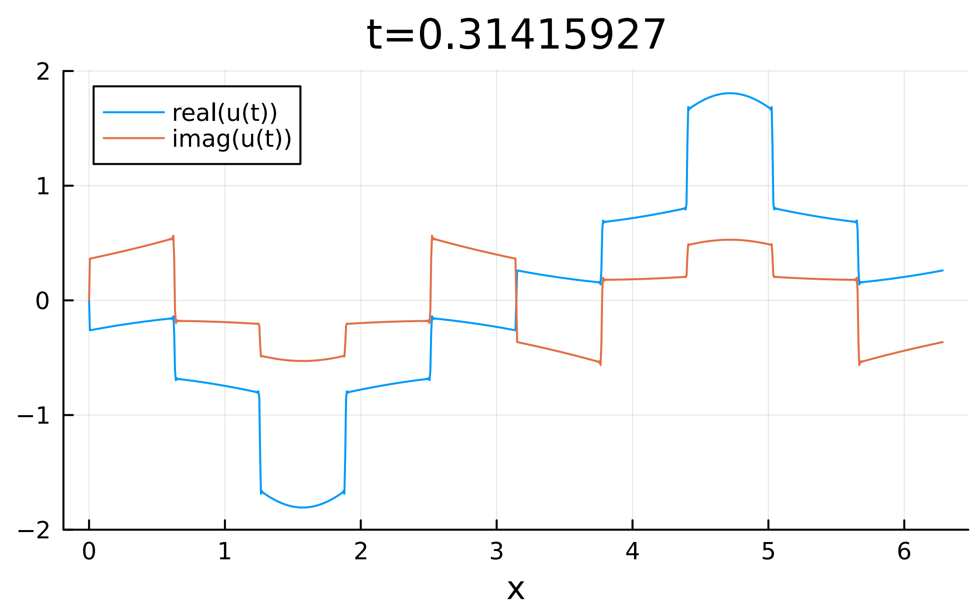
In [3]:

```
#times = range(0.,t_end,length=101)

p = NLS_ODE_cache_init(W0)
prob = ODEProblem(NLS_ODE!,W0,tspan,p)

solve(prob,DP5(),abstol=1e-3,reltol=1e-3, # This is to force compilin}g
      save_everystep=false)               # in order to measure pure run time

tol_1 = 1e-6

solDP5_1 = solve(prob,DP5(),abstol=tol_1,reltol=tol_1,save_everystep=false);
```

In [16]:

```
theta = omega*t_end

W_end_1= solDP5_1[end]
U_end_1 = expA(W_end_1,theta,p)


yrange=(-2epsilon,2epsilon)
xx = range(0,stop=2pi,length=J2)

plot(xlabel="x", title="t=$(Float32(t_end))", ylims=yrange, size=(500,300))
plot!(xx,real(U_end_1),label="real(u(t))")
plot!(xx,imag(U_end_1),label="imag(u(t))")
```

Out[16]:



We next solve it twice with a tighter tolerance, once with the original spatial semidiscretization, and once with a finer one. This will allow us to estimate the time-discretization error and the full discretization error.

In [5]:

```julia
tol_2 = tol_1/10

solDP5_2 = solve(prob,DP5(),abstol=tol_2,reltol=tol_2,save_everystep=false);
```

In [6]:

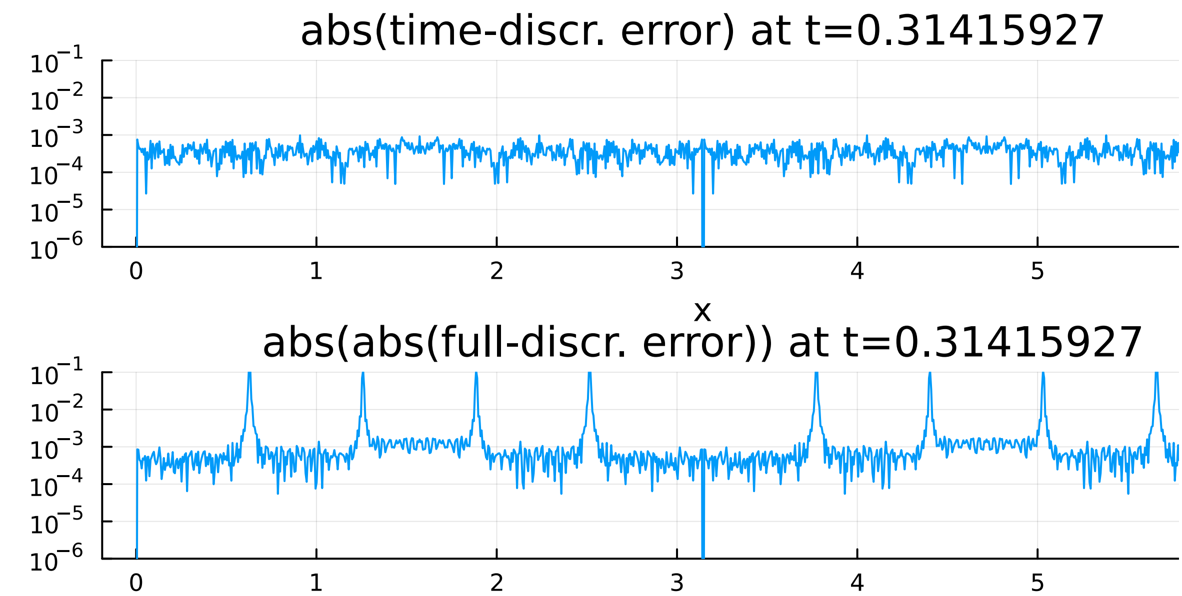
```julia

J_ = 2*J
J_2 = 2*J_


W0_=Vector{ComplexF64}(undef,J_2)
W0_[1]= zero(ComplexF64)
for i in 2:J_
    W0_[i]=-epsilon+0.0im
end
W0_[J_+1]= zero(ComplexF64)
for i in J_+2:J_2
    W0_[i]=epsilon+0.0im
end

p_ = NLS_ODE_cache_init(W0_)

prob_ = ODEProblem(NLS_ODE!,W0_,tspan,p_)

@time solDP5_ = solve(prob_,DP5(),abstol=tol_2,reltol=tol_2,save_everystep=fal

```

```
5.190759 seconds (85 allocations: 517.094 KiB)
```

In [7]:

```
W_end_1= solDP5_1[end]
W_end_2 = solDP5_2[end]
W_end_ = solDP5_[end]
U_end_1 = expA(W_end_1,theta,p)
U_end_2 = expA(W_end_2,theta,p)
U_end_ = expA(W_end_,theta,p_)

errU_t = U_end_1 - U_end_2 .+ eps()


xx = range(0,stop=2pi,length=J2)
yrange = (1e-6,1e-1)

pl_err_t = plot(xx,abs.(errU_t),yscale=:log10, ylims = yrange, xlabel="x",
                legend=false, title="abs(time-discr. error) at t=$(Float32(t_e

errU = U_end_1 - U_end_[1:2:end] .+ eps()

pl_err = plot(xx,abs.(errU), yscale=:log10, ylims = yrange,
              legend=false, title="abs(abs(full-discr. error)) at t=$(Float32(

plot(pl_err_t, pl_err,  layout=(2,1), size=(660,300))
```

Out[7]:



L2-norm of the two type of errors:

In [8]:

```
(norm(errU_t)/sqrt(2J), norm(errU)/sqrt(2J))
```

Out[8]:

```
(0.0004196552343838656, 0.0163397123247182)
```

Clearly, for the considered spatial semi-diskretization, it does not make much sense to use a tighter

Numerical solution with Taylor-Fourier

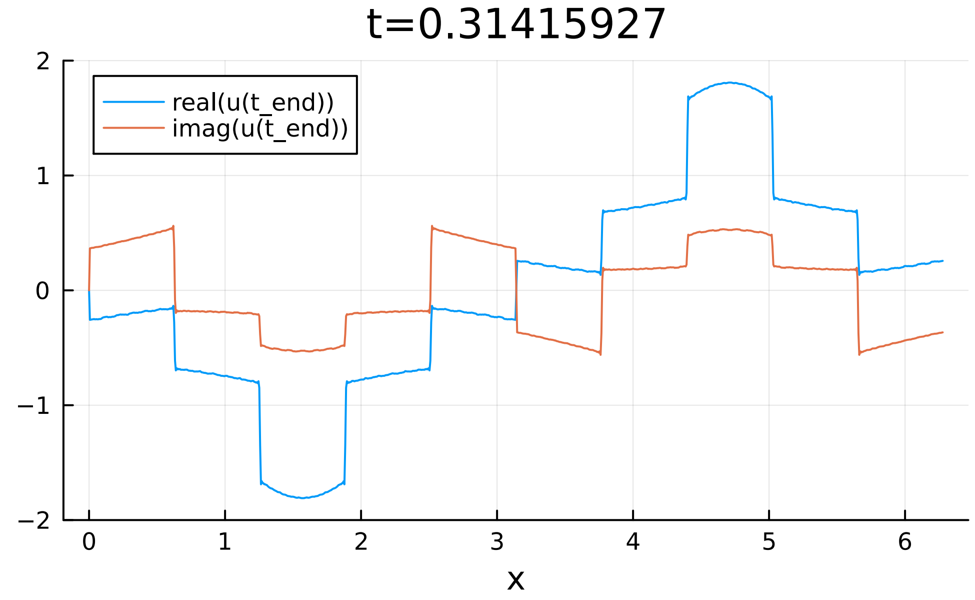
In [9]:

```
deg = 4
M = 4J

tf_cache = NLS_ODE_TF_cache_init(W0,omega,deg,M)

prob_TF = PeriodicODEProblem(NLS_ODE_TF!, tf_cache, W0, omega)

sol_TF = TaylorFourierSolve(prob_TF,deg,M);
```

In [17]:

```
W_TF = sol_TF(t_end)
theta = p.omega*(t_end)
U_TF = expA(W_TF,theta,p)


yrange=(-2epsilon,2epsilon)
xx = range(0,step=pi/J,length=J2)
plot(xlabel="x", title="t=$(Float32(t_end))", ylims=yrange, size=(500,300))
plot!(xx,real(U_TF),label="real(u(t_end))")
plot!(xx,imag(U_TF),label="imag(u(t_end))")
```

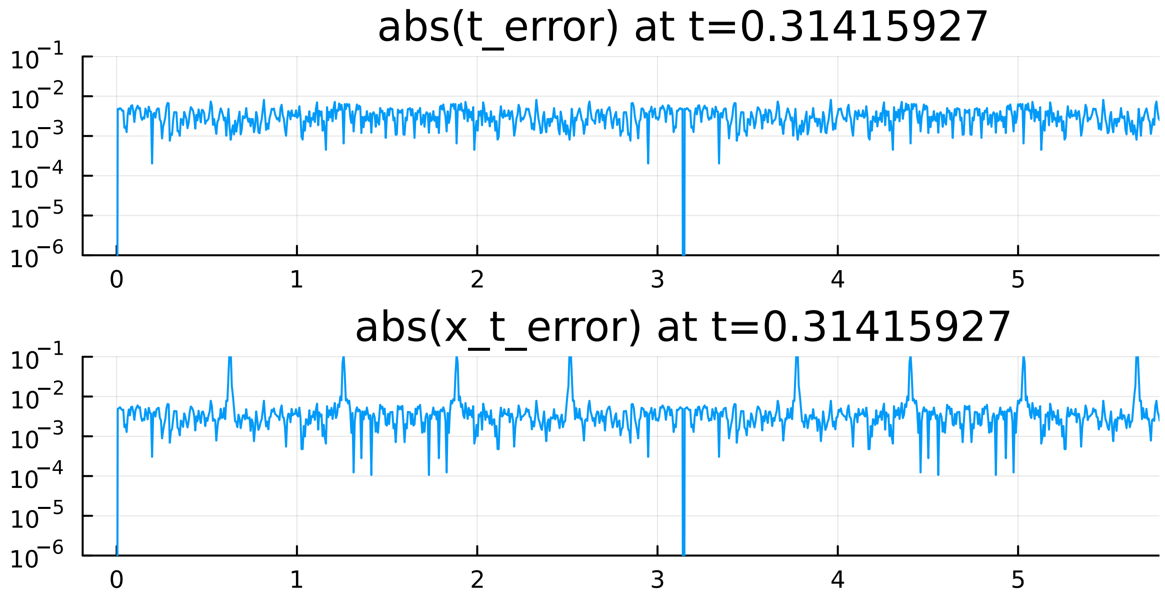
Out[17]:



In [11]:

```
errU_t = U_TF - U_end_2 .+ eps()


xx = range(0,stop=2pi,length=J2)
yrange = (1e-6,1e-1)

pl_err_t = plot(xx,abs.(errU_t),yscale=:log10, ylims = yrange,
                legend=false, title="abs(t_error) at t=$(Float32(t_end))")

errU = U_TF - U_end_[1:2:end] .+ eps()

pl_err = plot(xx,abs.(errU), yscale=:log10, ylims = yrange,
              legend=false, title="abs(x_t_error) at t=$(Float32(t_end))")

plot(pl_err_t, pl_err,  layout=(2,1), size=(660,300))
```

Out[11]:



In [12]:

```
(norm(errU_t)/sqrt(2J), norm(errU)/sqrt(2J))
```

Out[12]:

```
(0.003631823713845818, 0.016704579298438056)
```

Get get similar precision with both methods.

Let us now compare the CPU time of each integration:

In [13]:

```
@time solve(prob,DP5(),abstol=tol_1,reltol=tol_1,save_everystep=false);
@time TaylorFourierSolve(prob_TF,deg,M);

```

```
  0.412025 seconds (69 allocations: 262.344 KiB)
 16.395724 seconds (226 allocations: 2.750 GiB)
```

Our Taylor-Fourier integration gives, compared to the DP5-Lawson method, similar precision, but requires much more CPU time.

In [ ]:

```

```