```julia
begin
    import Pkg
    Pkg.activate(".")
    using SpinModels, SpinTruncatedWigner, OrdinaryDiffEq, Statistics, CairoMakie, LinearAlgebra,SparseArrays
end
```

Setup

staggered_magnetization (generic function with 1 method)

```julia
function staggered_magnetization(state; indices)
    odd = 1:2:length(indices)
    even = 2:2:length(indices)
    return (sum(state[indices[odd]]) - sum(state[indices[even]]))/length(indices)
end
```

N = 2

```julia
N = 2
```

hamiltonian (generic function with 1 method)

```julia
hamiltonian(Δ;N=N) = NN(Chain(2)) * XXZ(Δ)
```

psi0 = SpinProductState([SpinHalf(0, 0), SpinHalf(π, 0)])

```julia
psi0 = NeelState(N, :z)
```

times = 0.0:0.010025062656641603:4.0

```julia
times = range(0, 4; length=400)
```

dTWA

dtwa_exactsolution (generic function with 1 method)

```julia
# exact in the sense that the sampling is exact
# i.e. all possible initial states are sampled once
function dtwa_exactsolution(H, times, initial_states; int=Vern8(), abstol=1e-10,reltol=1e-10)
    prob = ODEProblem{true, SciMLBase.FullSpecialize}(
        SpinTruncatedWigner.twaUpdate!,
        zeros(3N),
        (0, maximum(times)),
        TWAParameters(H); saveat=sort(times))
    ensemble = EnsembleProblem(prob;
        prob_func = (prob, i, repeat) -> remake(prob; u0 = initial_states[i]))
    solve(ensemble, int; abstol, reltol, trajectories = length(initial_states))
end
```

dtwa_state_up_z =
[[-1.0, -1.0, 1.0], [-1.0, 1.0, 1.0], [1.0, -1.0, 1.0], [1.0, 1.0, 1.0]]

```
dtwa_state_up_z = [[x,y,1.0] for x in (-1,1) for y in (-1,1)]
```

dtwa_state_down_z = [[-1, -1, -1], [-1, 1, -1], [1, -1, -1], [1, 1, -1]]

```
dtwa_state_down_z = [[x,y,-1] for x in (-1,1) for y in (-1,1)]
```

dtwa_initial_states =
[[-1.0, -1.0, 1.0, -1.0, -1.0, -1.0], [-1.0, -1.0, 1.0, -1.0, 1.0, -1.0], [-1.0, -1.0, 1

```
dtwa_initial_states = [vcat(s1,s2) for s1 in dtwa_state_up_z for s2 in
dtwa_state_down_z]
# we sample the states exactly i.e. each phase-point vector exactly once
```

Δlist = [0, 2, 4, 6]

```
Δlist = [0,2,4,6]
```

dtwa_sols =
[EnsembleSolution Solution of length 16 with uType:
ODESolution{Float64, 2, Vector{Vector{Float64}}, Nothing, Nothing, Vector{Float64},

```
dtwa_sols = [dtwa_exactsolution(hamiltonian(Δ), times, dtwa_initial_states) for
Δ in Δlist]
```

dtwa_magnetization_indices = 3:3:6

```
dtwa_magnetization_indices = 3:3:3N
```

dtwa_results =
[[1.0, 0.996787, 0.987177, 0.971265, 0.949205, 0.921213, 0.887567, 0.8486, 0.804704,

```
dtwa_results = let f(sol) = mean(s->staggered_magnetization(s;
indices=dtwa_magnetization_indices), sol)
    [f.(sol.(times)) for sol in dtwa_sols]
end
```

ED

```
ed_result =
  [1.0, 0.996786, 0.987163, 0.971195, 0.948983, 0.92067, 0.886439, 0.846509, 0.801137, 0.
```

```julia
ed_result = let (evals, U) = eigen(Hermitian(Matrix(hamiltonian(2)))),
    D = Diagonal(evals),
    O = sparse(Z((-1) .^ (0:N-1))/2),
    # O = sparse(X(ones(N))),
    Uψ0 = U'*SpinTruncatedWigner.quantum(psi0),
    res = zeros(length(times))
    for (i,t) in enumerate(times)
        ψt = U*(cis(-D*t)*Uψ0)
        res[i] = real(dot(ψt, O, ψt))
    end
    res
end
```

Plot

```
save_and_display (generic function with 3 methods)
```

```julia
begin
    function save_and_display(name, folder="../plots")
        return fig -> save_and_display(name, fig, folder)
    end
    function save_and_display(name, fig, folder)
        mkpath(folder)
        mkpath(joinpath(folder, "png"))
        Makie.save(joinpath(folder, name*".pdf"), fig)
        Makie.save(joinpath(folder, "png", name*".png"), fig)
        fig
    end
end
```

```
HEIGHT = 250
```

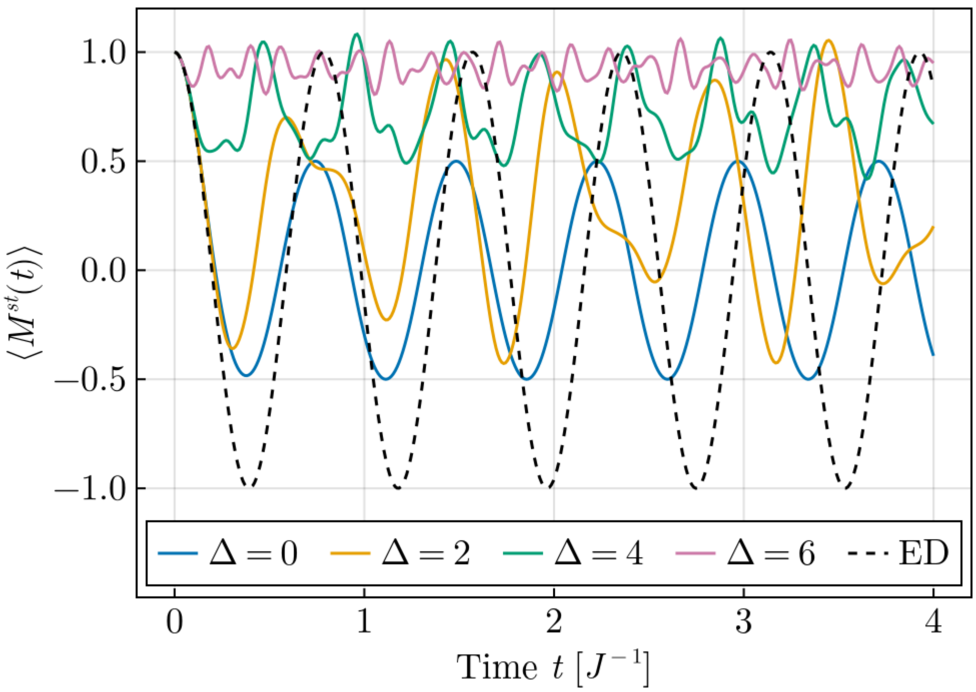
```julia
HEIGHT = 250
```

```
SCALE = 2
```

```julia
SCALE = 2
```

```
THEME =
Attributes with 7 entries:
  Axis => Attributes with 3 entries:
    xscale => identity
    xtickalign => 1
    ytickalign => 1
  figure_padding => (1, 7, 1, 1)
  fonts => Attributes with 4 entries:
    bold => FTFont (family = NewComputerModern, style = 10 Bold)
    bolditalic => FTFont (family = NewComputerModern, style = 10 Bold Italic)
    italic => FTFont (family = NewComputerModern, style = 10 Italic)
    regular => FTFont (family = NewComputerModern Math, style = Regular)
  fontsize => 18
  Label => Attributes with 3 entries:
    font => bold
    halign => left
    valign => top
  pt_per_unit => 0.5
  size => (492, 500)
```

```julia
THEME = merge(theme_latexfonts(),
    Theme(fontsize=9*SCALE, size=(246*SCALE,HEIGHT*SCALE), pt_per_unit=1/SCALE,
        figure_padding=(1,7,1,1),
        Axis=(; xtickalign=1, ytickalign=1, xscale=identity),
        Label=(; font=:bold,
            halign=:left, valign=:top)))
```




```julia
with_theme(THEME) do
    let fig = Figure(;size=(246*SCALE,170*SCALE)),
        ax = Axis(fig[1,1]; ylabel=L"\langle M^{st}(t)\rangle", xlabel=L"Time $t$ [$J^{-1}$]")
        for (Δ, res) in zip(Δlist, dtwa_results)
            lines!(ax, times, res; label=L"\Delta=%$(Δ)")
        end
        # lines!(ax, times, gctwa_result; label="gcTWA")
        # lines!(ax, times, dctwa_result; label="dcTWA")
        lines!(ax, times, ed_result; label="ED", color=:black, linestyle=:dash)
        axislegend(ax; orientation=:horizontal, position=:cb)
        #xlims!(ax, extrema(times)...)
        ylims!(ax, -1.5,1.2)
        ax.yticks = -1:0.5:1
        fig
    end
end |> save_and_display("appendix-two-spin-dynamics")
```