

Van der Pol 振動子

- Author: 黒木玄
- Date: 2019-04-05～2019-04-15
- Repository: <https://github.com/genkuroki/DifferentialEquations> (<https://github.com/genkuroki/DifferentialEquations>)

このファイルは [nbviewer \(https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/01-1%20Van%20der%20Pol%20oscillator.ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/01-1%20Van%20der%20Pol%20oscillator.ipynb) でも閲覧できる。

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) と [Jupyter環境 \(https://jupyter.org/\)](https://jupyter.org/) の簡単な解説については次を参照せよ:

- [JuliaとJupyterのすすめ \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true)

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) 環境の整備の仕方については次を参照せよ:

- [Julia v1.1.0 の Windows 8.1 へのインストール \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb)

目次

[1 Van der Pol 振動子](#)

[2 Van der Pol 方程式のベクトル場と流れの図](#)

[2.1 \$\mu = 0\$ の場合](#)

[2.2 \$\mu\$ が正の場合](#)

[2.3 \$\mu\$ が負の場合](#)

[3 Van der Pol 方程式の数値解](#)

[3.1 \$\mu = 0\$](#)

[3.2 \$\mu = 0.1\$](#)

[3.3 \$\mu = 0.5\$](#)

[3.4 \$\mu = 1\$](#)

[4 正弦波による強制力付きの場合](#)

In [1]:

```
1  using PyPlot: PyPlot, plt
```

In [2]:

```

1 ▼ function plot_stream(f, g;
2     x = range(-4, 4, length=201),
3     y = range(-4, 4, length=201),
4     density = 1.2, figtitle="", sign=1.0)
5
6     # meshgrid
7     xx = repeat(x', length(y), 1)
8     yy = repeat(y, 1, length(x))
9     XX = f.(xx, sign*yy)
10    YY = g.(xx, sign*yy) * sign
11
12    plt.streamplot(xx, yy, XX, YY, linewidth=0.5, density=density, color="blue")
13    plt.xlim(extrema(x)...)
14    plt.ylim(extrema(y)...)
15    plt.grid(ls=":")
16    figtitle == "" || plt.title(figtitle)
17 end
18
19 ▼ function plot_vector_field(f, g;
20     x = range(-4, 4, length=21),
21     y = range(-4, 4, length=21),
22     figtitle="", scale=2.0, sign=1.0)
23
24     # meshgrid
25     xx = repeat(x', length(y), 1)
26     yy = repeat(y, 1, length(x))
27     XX = f.(xx, sign*yy)
28     YY = g.(xx, sign*yy) * sign
29     MM = maximum(@.(sqrt(XX^2+YY^2)))
30     #println(MM) # for debug
31
32     plt.quiver(xx, yy, XX, YY, scale=scale*MM, color="red")
33     plt.xlim(extrema(x)...)
34     plt.ylim(extrema(y)...)
35     plt.grid(ls=":")
36     figtitle == "" || plt.title(figtitle)
37 end

```

Out[2]: plot_vector_field (generic function with 1 method)

1 Van der Pol 振動子

[Van der Pol \(ファン・デル・ポル\) 振動子](https://en.wikipedia.org/wiki/Van_der_Pol_oscillator) (https://en.wikipedia.org/wiki/Van_der_Pol_oscillator) とは以下の非線形常微分方程式で記述される系のことである:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0.$$

ここで $x = x(t)$ の時刻 t に関する1階と2階の導函数を \dot{x} , \ddot{x} と表した. この微分方程式を **Van der Pol 方程式** と呼ぶ.

2 Van der Pol 方程式のベクトル場と流れの図

$v = \dot{x}$ とおくと, Van der Pol 方程式は次の連立常微分方程式に書き直される:

$$\begin{cases} \dot{x} = v, \\ \dot{v} = \mu(1 - x^2)v - x. \end{cases}$$

この微分方程式は位置 x と速度 v を平面上の点 (x, v) で表すとき, その点の動きの速度ベクトルが $(v, \mu(1 - x^2)v - x)$ となることを意味している. 平面上の各点にその点における速度ベクトルを対応させる函数を平面上の **ベクトル場** と呼ぶ. 上の連立常微分方程式はベクトル場に沿って平面上の点が流れて行く様子を表している.

In [3]:

```

1 ▼ function plot_Van_der_Pol(μ; scale=1/abs(μ), sign=1.0)
2     # Van der Pol equation
3     f(x,v) = v
4     g(x,v) = μ*(1-x^2)*v - x
5
6     plt.figure(figsize=(7,3.5))
7     plt.subplot("121")
8     plot_stream(f, g, figtitle="Van der Pol: μ = $μ", sign=sign)
9     plt.subplot("122")
10    plot_vector_field(f, g, figtitle="Van der Pol: μ = $μ", scale=scale, sign=sign)
11    plt.tight_layout()
12 end

```

Out[3]: plot_Van_der_Pol (generic function with 1 method)

2.1 $\mu = 0$ の場合

$\mu = 0$ のとき, Van der Pol 方程式は

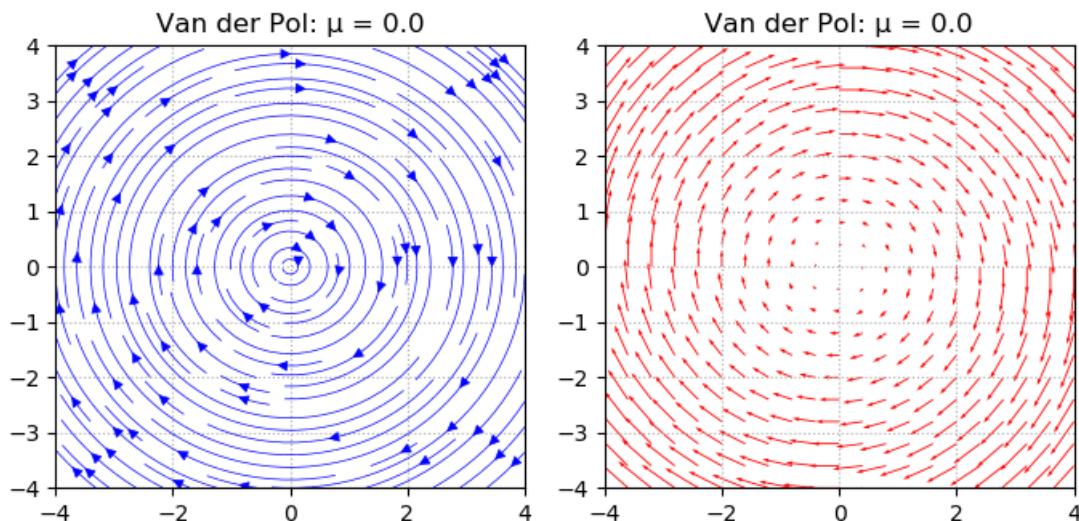
$$\begin{cases} \dot{x} = v, \\ \dot{v} = -x. \end{cases}$$

の形になる. これが定める流れは時計回りで回転する流れになる. その一般解は, $r \geq 0$ と実数 a を使って次のように書ける:

$$\begin{cases} \dot{x} = r \sin(t - a), \\ \dot{v} = r \cos(t - a). \end{cases}$$

In [4]:

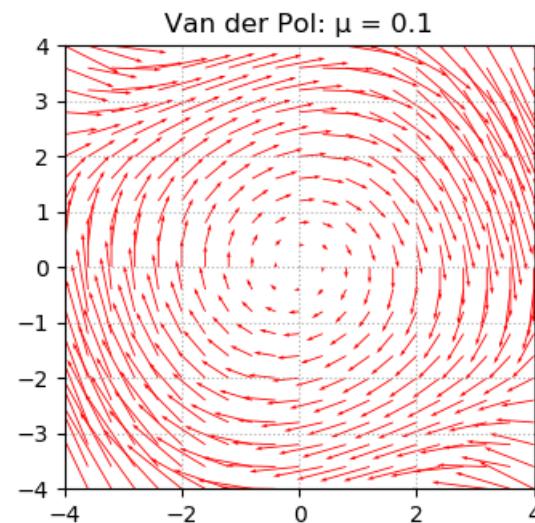
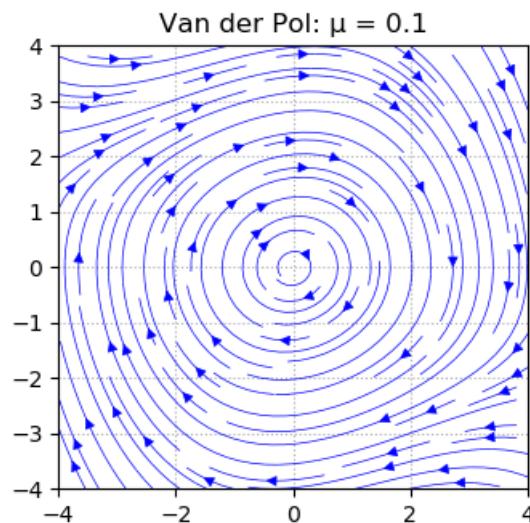
1 plot_Van_der_Pol(0.0, scale=10.0)



2.2 μ が正の場合

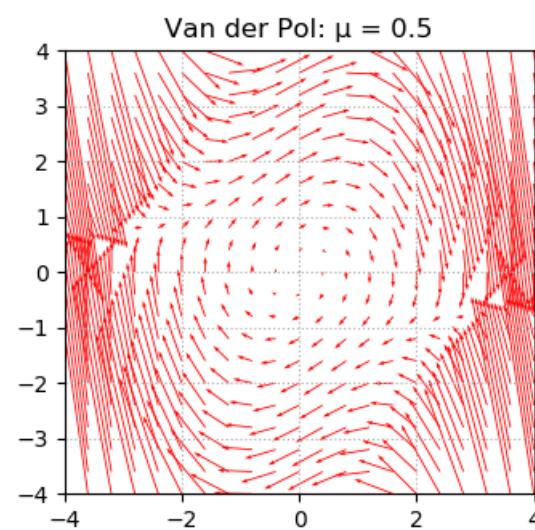
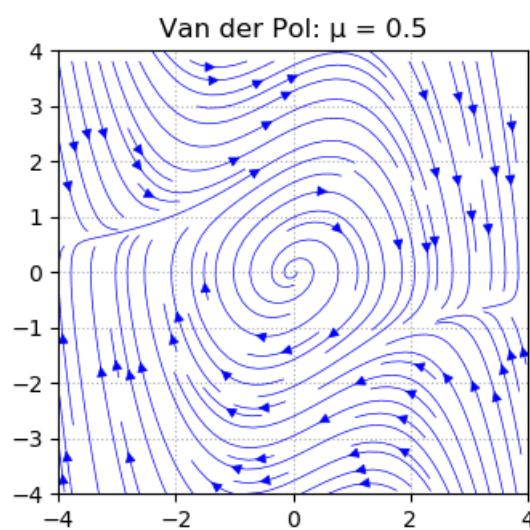
In [5]:

1 plot_Van_der_Pol(0.1)



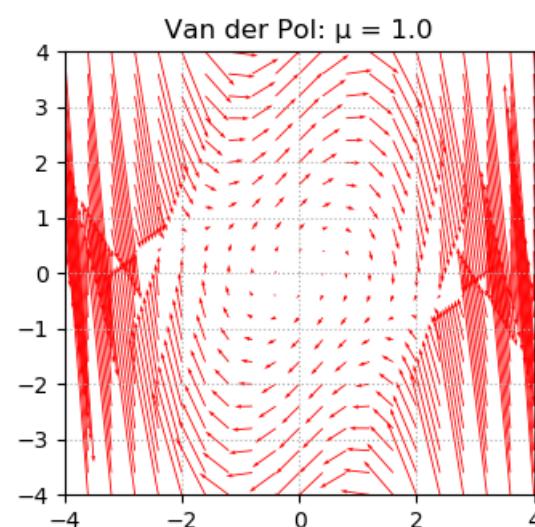
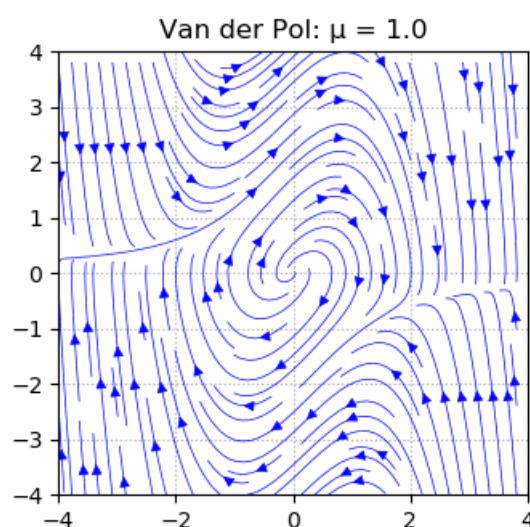
In [6]:

1 plot_Van_der_Pol(0.5)



In [7]:

1 plot_Van_der_Pol(1.0)



2.3 μ が負の場合

Van der Pol 方程式

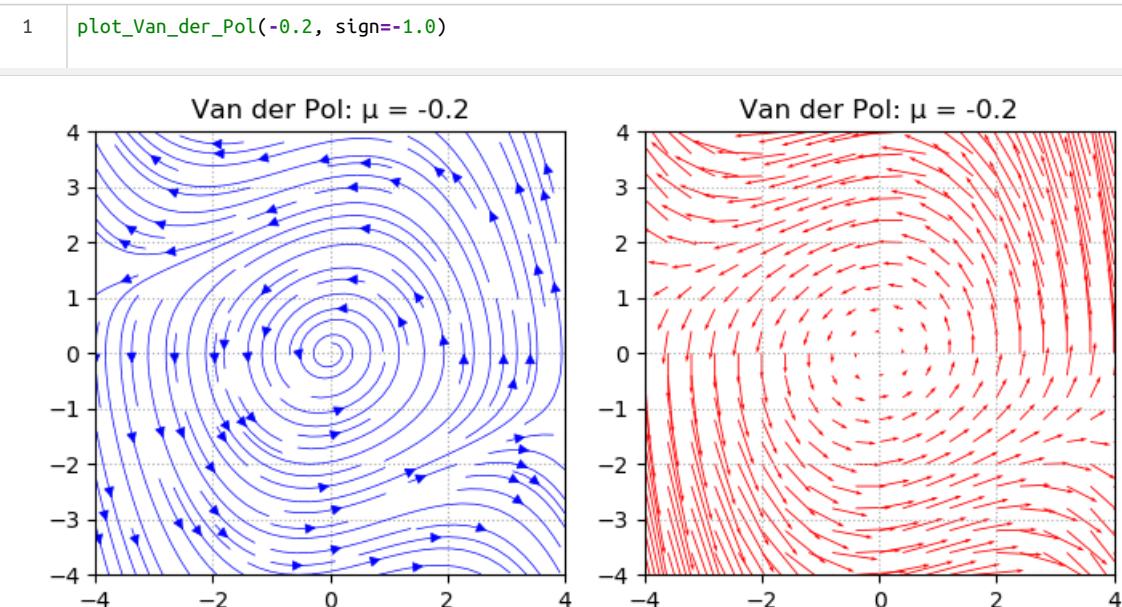
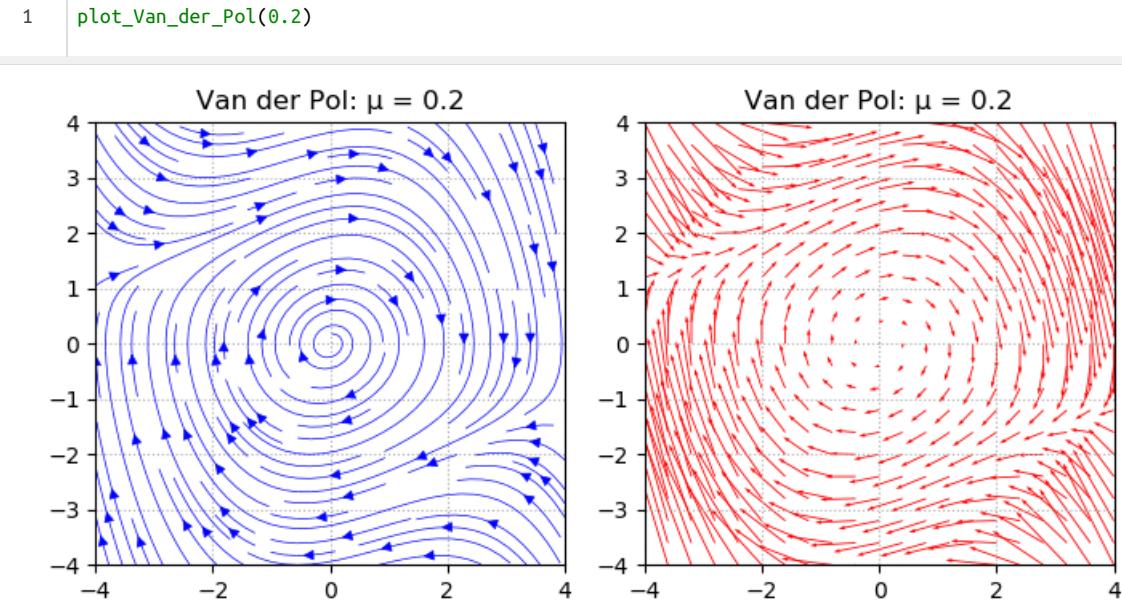
$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0.$$

で時刻 t が進む向きを逆転させると、すなわち、 t に $-t$ を代入すると、

$$\ddot{x} + \mu(1 - x^2)\dot{x} + x = 0.$$

の形になる。すなわち、時間の向きを逆転させることと、パラメーター μ をその -1 倍で置き換えることは等しい。

以下の $\mu = 0.2$ と $\mu = -0.2$ の場合のプロットを比較すると、矢印の向きが逆になっていることがわかる。



注意: 上のセルの `sign=-1.0` はプロットするときに縦軸(v 軸)の向きを反転してプロットすることを意味している。

3 Van der Pol 方程式の数値解

In [10]:

```

1 using DifferentialEquations
2 using Plots
3 default(:blegend, plot_color(default(:bg), 0.7))
4 default(:flegend, plot_color(ifelse(isdark(plot_color(default(:bg))), :white, :black), 0.6))
5
6 gr()
7 ENV["PLOTS_TEST"] = "true";

```

In [11]:

```

1 ▼ ##### Van der Pol 方程式の記述
2 ▼ function vanderpol!(du, u, p, t)
3     # u[1] = x
4     # u[2] = v
5     # du[1] = dx/dt
6     # du[2] = dv/dt
7     # p[1] = μ
8
9     # dx/dt = v
10    du[1] = u[2]
11    # dv/dt = μ (1 - x^2) v - x
12    du[2] = p[1]*(1 - u[1]^2)*u[2] - u[1]
13 end
14
15 ▼ function solve_vanderpol(μ, x₀, v₀, tmax)
16     u₀ = [x₀, v₀] # initial values
17     p = [μ] # parameters
18     tspan = (0.0, tmax) # time span
19     prob = ODEProblem(vanderpol!, u₀, tspan, p) # ordinary differential equation problem
20     solve(prob)
21 end
22
23 ▼ function plot_vanderpol_sol(sol, μ; l1=:best, l2=:best)
24     plot(size=(600, 250))
25     plot!(sol, label=["x", "v"], legend=l1, lw=1)
26     title!("Van der Pol equation (μ = $μ)", titlefontsize=12) !> display
27
28     plot(size=(300, 300))
29     plot!(sol, vars=(1,2), label="(x,v)", legend=l2, lw=1)
30     title!("Van der Pol equation (μ = $μ)", titlefontsize=12) !> display
31 end
32
33 ▼ function solve_and_plot_vanderpol(; μ=0.5, x₀=0.1, v₀=0.0, tmax=50.0, l1=:topleft, l2=:best)
34     sol = solve_vanderpol(μ, x₀, v₀, tmax)
35     plot_vanderpol_sol(sol, μ; l1=l1, l2=l2)
36 end

```

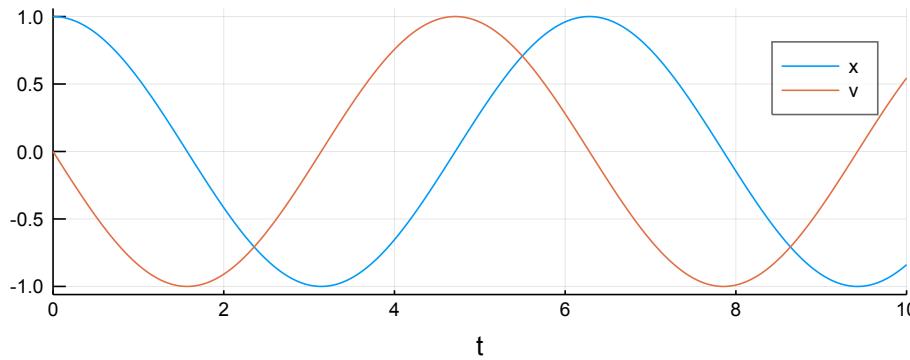
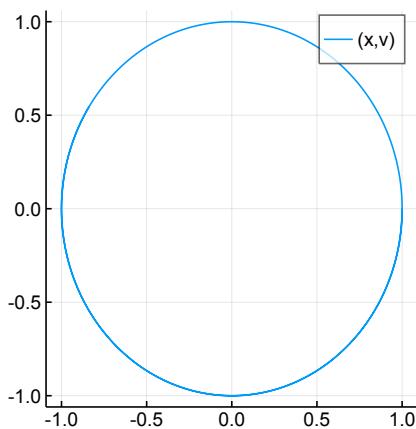
Out[11]: solve_and_plot_vanderpol (generic function with 1 method)

3.1 $\mu = 0$

$\mu = 0$ の場合に van der Pol 方程式の解は (x, v) 平面上の円運動になる。

In [12]:

```
1 solve_and_plot_vanderpol(; mu=0.0, x0=1.0, v0=0.0, tmax=10.0, l1=:topright)
```

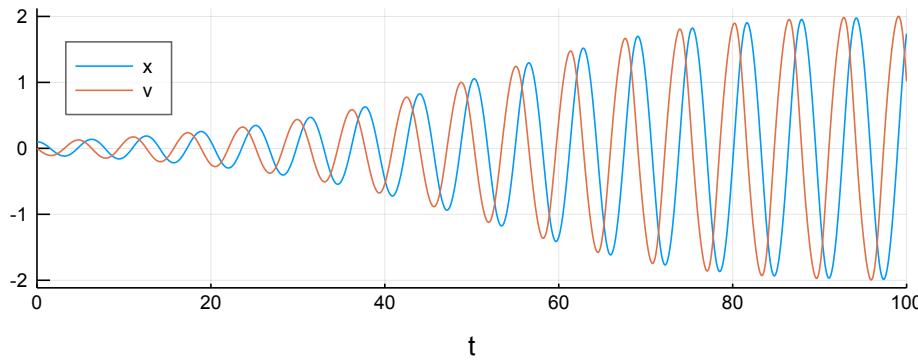
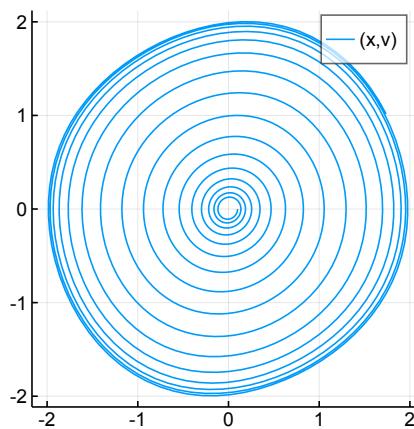
Van der Pol equation ($\mu = 0.0$)Van der Pol equation ($\mu = 0.0$)

3.2 $\mu = 0.1$

$\mu = 0.1$ の場合に, van der Pol 方程式の解は (x, v) 平面上の歪んだ円に内側と外側から巻き付くような運動になる. 巻き付く先の歪んだ円は **limit cycle** と呼ばれる.

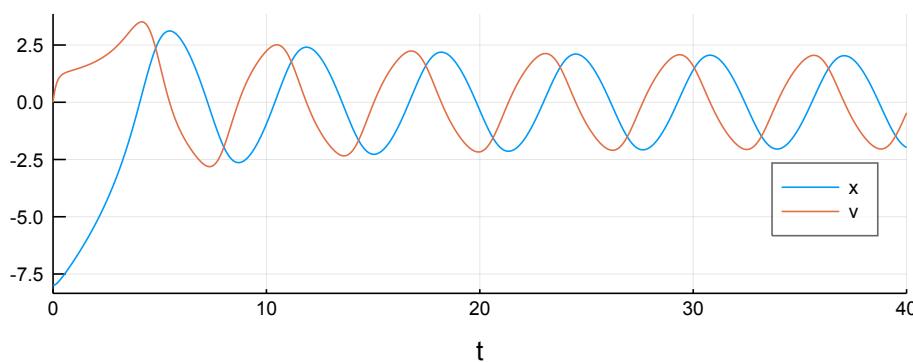
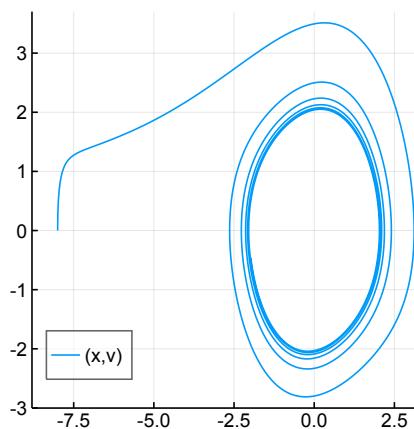
In [13]:

1 solve_and_plot_vanderpol(; mu=0.1, x0=0.1, v0=0.0, tmax=100.0)

Van der Pol equation ($\mu = 0.1$)Van der Pol equation ($\mu = 0.1$)

In [14]:

1 solve_and_plot_vanderpol(; mu=0.1, x0=-8.0, v0=0.0, tmax=40.0, l1=:bottomright, l2=:bottomleft)

Van der Pol equation ($\mu = 0.1$)Van der Pol equation ($\mu = 0.1$)

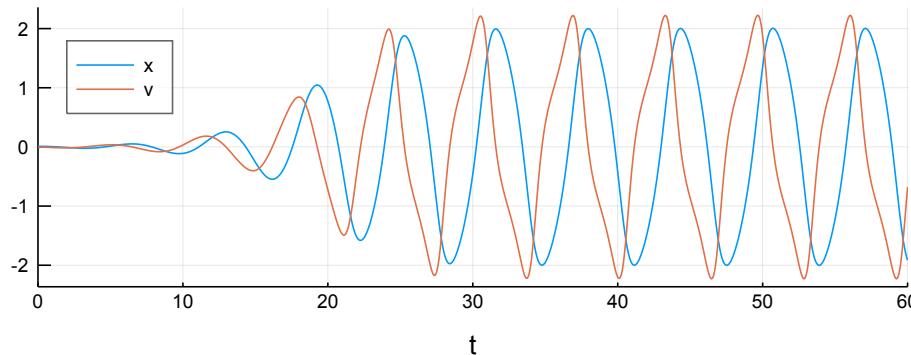
3.3 $\mu = 0.5$

μ を大きくすると limit cycle の形が円から離れて行く。

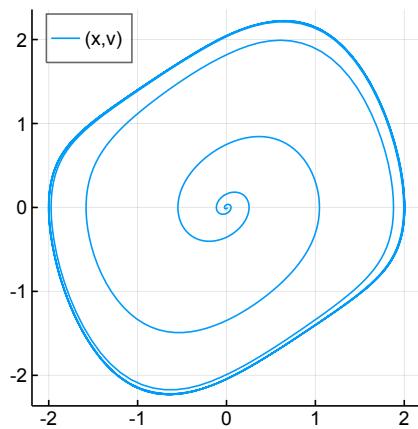
In [15]:

```
1 solve_and_plot_vanderpol(; mu=0.5, x0=0.01, v0=0.0, tmax=60.0, l2=:topleft)
```

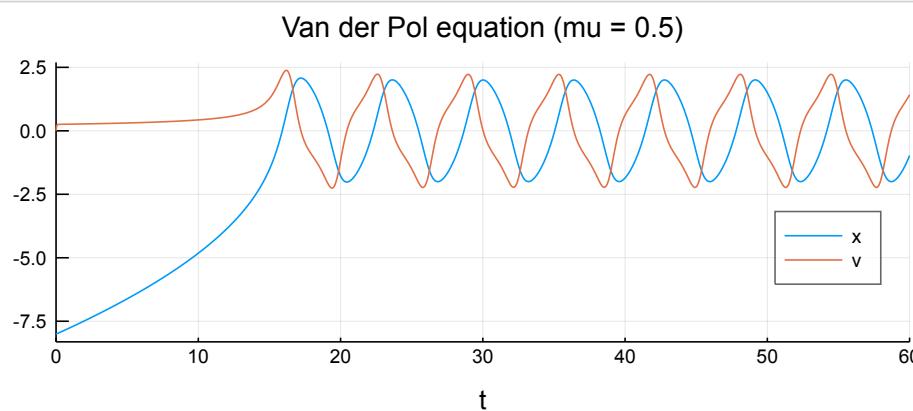
Van der Pol equation ($\mu = 0.5$)



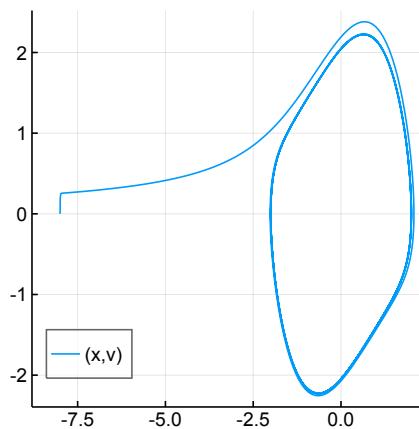
Van der Pol equation ($\mu = 0.5$)



In [16]: 1 solve_and_plot_vanderpol(; mu=0.5, x0=-8.0, v0=0.0, tmax=60.0, l1=:bottomright, l2=:bottomleft)



Van der Pol equation ($\mu = 0.5$)

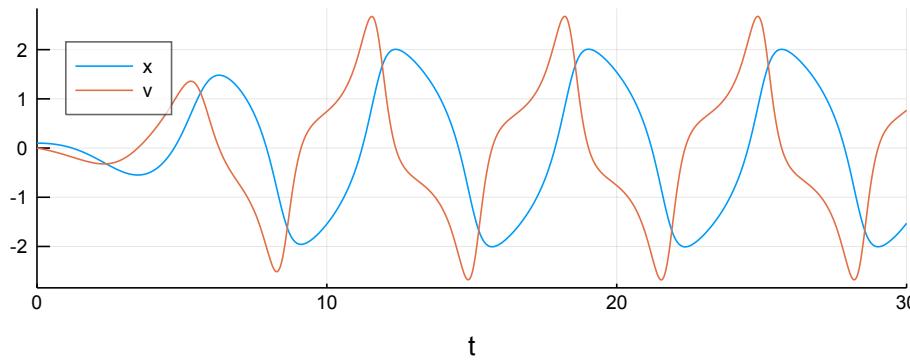


3.4 $\mu = 1$

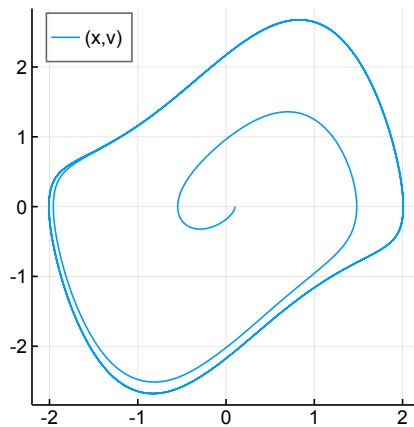
$\mu = 1$ での limit cycle は斜めに傾いた平行四辺形に近い形になる。

In [17]: 1 solve_and_plot_vanderpol(; mu=1.0, x0=0.1, v0=0.0, tmax=30.0, l2=:topleft)

Van der Pol equation (mu = 1.0)

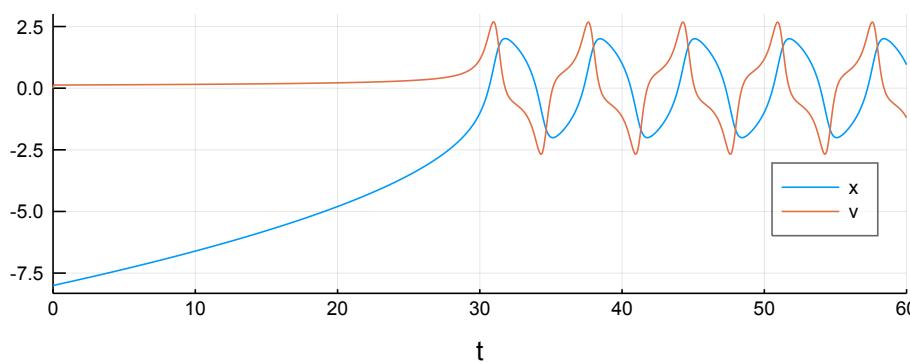


Van der Pol equation (mu = 1.0)

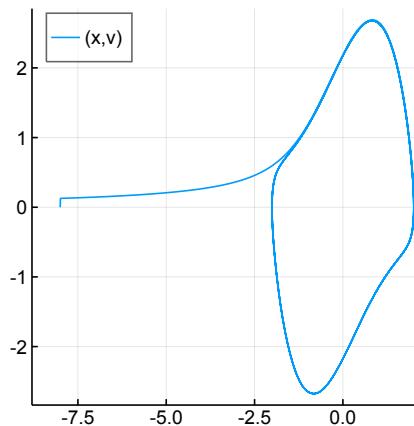


In [18]: 1 solve_and_plot_vanderpol(; mu=1.0, x0=-8.0, v0=0.0, tmax=60.0, l1=:bottomright, l2=:topleft)

Van der Pol equation (mu = 1.0)



Van der Pol equation (mu = 1.0)



4 正弦波による強制力付きの場合

$$\begin{cases} \dot{x} = v, \\ \dot{v} = \mu(1 - x^2)v - x + A \sin \omega t. \end{cases}$$

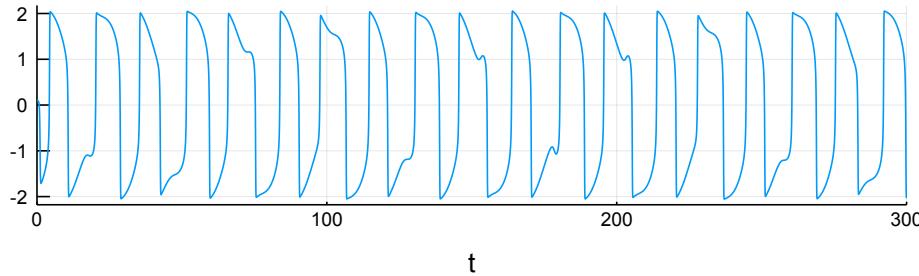
```
In [19]: 1 using DifferentialEquations
2 using Plots
3 default(:bglegend, plot_color(default(:bg), 0.7))
4 default(:fglegend, plot_color(ifelse(isdark(plot_color(default(:bg))), :white, :black), 0.6))
5
6 gr()
7 ENV["PLOTS_TEST"] = "true";
```

```
In [20]: 1 # sin による強制力付きの Van der Pol 方程式の記述
2 function vanderpolsin!(du, u, p, t)
3     # x = u[1]
4     # v = u[2]
5     # dx/dt = du[1]
6     # dv/dt = du[2]
7     # μ = p[1]
8     # A = p[2]
9     # ω = p[3]
10
11    # dx/dt = v
12    du[1] = u[2]
13    # dv/dt = μ(1 - x^2) v - x + A sin ωt
14    du[2] = p[1]*(1 - u[1]^2)*u[2] - u[1] + p[2]*sin(p[3]*t)
15 end
16
17 function solve_vanderpolsin(; μ=1.0, A=1.2, ω=2π/10, x0=0.1, v0=0.0, tmax=100)
18     u0 = [x0, v0] # initial values
19     p = [μ, A, ω] # parameters
20     tspan = (0.0, tmax) # time span
21     prob = ODEProblem(vanderpolsin!, u0, tspan, p) # ordinary differential equation problem
22     solve(prob)
23 end
24
25 function plot_vanderpol_sol(sol, μ, A, ω; l1=:best, l2=:best)
26     plot(size=(600, 200))
27     plot!(sol, vars=(0,1), label="x", legend=l1, lw=1)
28     title!("Forced Van der Pol (μ = $μ, A = $A, ω = $(round(ω, digits=2)))", titlefontsize=10) |> display
29
30     sleep(0.1)
31     plot(size=(300, 300))
32     plot!(sol, vars=(1,2), label="(x,v)", legend=l2, lw=1)
33     title!("μ = $μ, A = $A, ω = $(round(ω, digits=2))", titlefontsize=10) |> display
34 end
35
36 function solve_and_plot_vanderpolsin(; μ=1.0, A=1.2, ω=2π/5, x0=0.1, v0=0.0, tmax=50.0, l1=false, l2=false)
37     sol = solve_vanderpolsin(μ=μ, A=A, ω=ω, x0=x0, v0=v0, tmax=tmax)
38     plot_vanderpol_sol(sol, μ, A, ω; l1=l1, l2=l2)
39 end
```

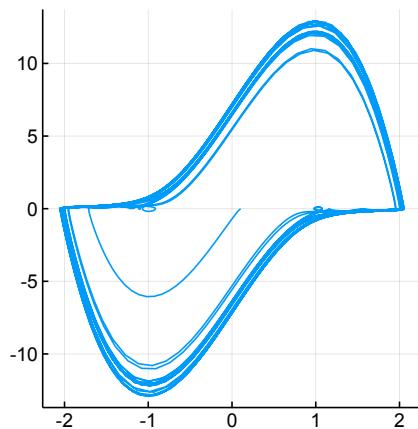
Out[20]: solve_and_plot_vanderpolsin (generic function with 1 method)

In [21]: 1 solve_and_plot_vanderpolsin(; mu=8.53, A=1.2, omega=2pi/10, x0=0.1, v0=0.0, tmax=300.0)

Forced Van der Pol (mu = 8.53, A = 1.2, omega = 0.63)



mu = 8.53, A = 1.2, omega = 0.63



カオス的な動き方をしている。

境界値問題 (線形常微分方程式の場合)

- Author: 黒木玄
- Date: 2019-04-15
- Repository: <https://github.com/genkuroki/DifferentialEquations> (<https://github.com/genkuroki/DifferentialEquations>)

このファイルは [nbviewer \(https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/02-1%20Boundary%20value%20problems%20\(linear%20ODE%20case\).ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/02-1%20Boundary%20value%20problems%20(linear%20ODE%20case).ipynb) でも閲覧できる。

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) と [Jupyter環境 \(https://jupyter.org/\)](https://jupyter.org/) の簡単な解説については次を参照せよ:

- [JuliaとJupyterのすすめ \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true)

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) 環境の整備の仕方については次を参照せよ:

- [Julia v1.1.0 の Windows 8.1 へのインストール \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb)

目次

1 微分方程式の分類
2 初期条件や境界条件
2.1 調和振動子
2.2 調和振動子の初期値問題
2.3 調和振動子のDirichlet型境界値問題
2.4 調和振動子のNeumann型境界値問題
3 BVPProblemの使い方
3.1 境界条件1
3.2 境界条件1(別の方)
3.3 境界条件2
4 調和振動子の数値解
4.1 初期値問題
4.2 齊次Dirichlet境界条件
4.3 齊次Neumann境界条件
4.4 非齊次Dirichlet境界条件
4.5 非齊次Neumann境界条件

```
In [1]: 1 using DifferentialEquations
         2 using Plots
         3 default(:bglegend, plot_color(default(:bg), 0.7))
         4 default(:fglegend, plot_color(ifelse(isdark(plot_color(default(:bg))), :white, :black), 0.2))
         5
         6 gr()
         7 ENV["PLOTS_TEST"] = "true";
```

1 微分方程式の分類

偏微分が出て来ない微分方程式を **常微分方程式 (ordinary differential equation)** と呼ぶ。

例. 調和振動子: $\omega > 0$ に対する微分方程式

$$\ddot{u} = -\omega^2 u$$

は常微分方程式である。□

例. 単振子: ω^2 に対する微分方程式

$$\ddot{u} = -\omega^2 \sin u$$

も常微分方程式である。□

偏微分が出て来る微分方程式を **偏微分方程式 (partial differential equation)** と呼ぶ。

例. 熱方程式: 偏微分方程式

$$u_t = u_{xx}$$

を空間次元1次元の **熱方程式 (heat equation)** と呼ぶ. □

例. 热方程式: $c > 0$ に対して, 偏微分方程式

$$\frac{1}{c^2} u_{tt} = u_{xx}$$

を空間次元1次元の **波動方程式 (heat equation)** と呼ぶ. □

例. Korteweg-de Vries 方程式: 偏微分方程式

$$u_t + u_{xxx} + 6uu_x = 0$$

を **KdV方程式** と呼ぶ. □

さらに, 微分方程式には線形と非線形の区別もあって, 調和振動子と熱方程式や波動方程式は線形であり, 单振子やKdV方程式は非線形の微分方程式である.

一般に unknown function u に関する線形作用素 P によって $Pu = 0$ と書ける微分方程式は **線形微分方程式 (linear differential equation)** もしくは **齊次線形微分方程式 (せいじ, homogeneous linear differential equation)** と呼ぶ. さらに与えられた函数 f のデータが追加されて, $Pu = f$ と書ける微分方程式を **非齊次線形微分方程式 (ひせいじ, inhomogeneous differential equation)** と呼ぶ

例えば, $P = (d/dt)^2 + \omega^2$ のとき, $Pu = \ddot{u} + \omega^2 u$ なので, $Pu = 0$ は調和振動子の微分方程式になる. 調和振動子の微分方程式は齊次線形常微分方程式である.

例えば, $P = (\frac{1}{c} \partial/\partial t)^2 - (\partial/\partial x)^2$ のとき $Pu = c^{-2} u_{tt} - u_{xx}$ なので $Pu = 0$ は波動方程式になる. 波動方程式は齊次線形偏微分方程式である.

非齊次な線形微分方程式の典型例は外部から強制力が与えられている場合に得られる微分方程式である. 調和振動子にが与えられているとき, 微分方程式は

$$\ddot{u} = -\omega^2 u + f(t)$$

の形になる. $f(t)$ が外力を表す. この微分方程式は $P = (d/dt)^2 + \omega^2$ とおくとき, $Pu = f$ と書ける. 外力が与えられた調和振動子は非齊次な線形常微分方程式である.

微分方程式が時間発展を意味するとき, 微分方程式中に時間変数 t が明示的に含まれているとき, その微分方程式は **非自励的 (non-autonomous)** と呼ばれる. 微分方程式中に時間変数 t が明示的に含まれないとき, その微分方程式は **自励的 (autonomous)** であるという. 自励的な微分方程式で記述される系は **自励系 (autonomous system)** と呼ばれ, 非自励的な微分方程式で記述される系は **非自励系 (non-autonomous system)** と呼ばれる.

例. Painlevé I and II: 非自励的な常微分方程式

$$\frac{du^2}{dt^2} = 6y^2 + t$$

は **Painlevé I 方程式 (the first Painlevé equation)** と呼ばれ, 非自励的な常微分方程式

$$\frac{du^2}{dt^2} = 2y^3 + ty + \alpha$$

は **Painlevé II 方程式 (the second Painlevé equation)** と呼ばれる. 他にも Painlevé III~VI 方程式があり, どれも2階の非自励的非線形常微分方程式である. □

微分方程式の階数は微分方程式が含む最高次の導函数の階数のことである.

例. 楕円函数の満たす微分方程式:

$$\left(\frac{dy}{dt} \right)^2 = 4y^3 + 2ay + b$$

の形の自励的な非線形常微分方程式の解は **椭円函数** と呼ばれるクラスの函数になることが知られている. この微分方程式の解として椭円函数を定義することもできる. この微分方程式の両辺を微分すると次の微分方程式が得られる:

$$2 \frac{dy}{dt} \frac{dy^2}{dt^2} = 12y^2 \frac{dy}{dt} + 2a \frac{dy}{dt}.$$

両辺を $2dy/dt$ で割ると

$$\frac{dy^2}{dt^2} = 6y^2 + a.$$

上の Painlevé I 方程式はこの橙円函数が満たす微分方程式における定数 a を時間変数 t に置き換えた形をしている. □

おまけ: 次のノートブックでは熱方程式, KdV方程式, Schrödinger方程式, Smith方程式を数値的に解いている:

- FFTを用いた偏微分方程式の数値解法(in-place版)
(<https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/A01%20Solving%20heat%2C%20KdV%2C%20Schr%20FFT.ipynb>)

非常に沢山の微分方程式があって、数値的に解くための様々な方法が存在する。それらをマスターするには自分で試行錯誤してみるのが一番早い。他人の話を聞くのではなく、自分でやってみることが重要である。

2 初期条件や境界条件

微分方程式を解くときには、与えられた微分方程式を満たすすべての解を求めるのではなく、目的に応じて適切な条件を満たす解を求めることが多い。

「目的に応じた適切な条件」として現われる条件の典型例が **初期条件** や **境界条件** である。

それらについて例を用いて説明したい。

2.1 調和振動子

$\omega > 0$ であると仮定する。次の微分方程式を調和振動子の微分方程式と呼ぶ:

$$\ddot{x} = -\omega^2 x.$$

この微分方程式の任意の解は定数 A, B によって次のように表わされる:

$$x(t) = A \cos \omega t + B \sin \omega t.$$

この式の右辺が調和振動子の微分方程式の解になっていることは時間変数 t で2回微分してみればすぐにわかる。任意の解がこのように表わされることは2階の線形常微分方程式の解空間(解全体のなすベクトル空間)が2次元になるという事実からわかる。

2.2 調和振動子の初期値問題

任意に与えられた定数 a, b に対して、調和振動子 $\ddot{x} = -\omega^2 x$ の解で初期条件

$$x(0) = a, \quad \dot{x}(0) = b\omega$$

を満たすものが唯一つ存在することを示そう。この初期条件に一般解

$$x(t) = A \cos \omega t + B \sin \omega t$$

を代入することによって、

$$A = a, B = b$$

を得る。これで

$$x(t) = a \cos \omega t + b \sin \omega t$$

が上の初期条件を満たす唯一つの解になる。

2.3 調和振動子のDirichlet型境界値問題

$T > 0$ と与えられた定数 a, b に対して、調和振動子 $\ddot{x} = -\omega^2 x$ の解で境界条件

$$x(0) = a, \quad x(T) = b$$

を満たすのをすべて求めてみよう。これに一般解 $x(t) = A \cos \omega t + B \sin \omega t$ を代入すると

$$A = a, \quad A \cos \omega T + B \sin \omega T = b.$$

A は $A = a$ によって一意的に決まるが、 B については $\sin \omega T$ が 0 であるか否かによって場合分けが必要になる。

$\sin \omega T \neq 0$ の場合. そのとき, 任意の a, b の組み合わせに対して, A, B が次のように一意的に決まる:

$$A = a, \quad B = \frac{b - a \cos \omega T}{\sin \omega T}.$$

すなわち,

$$x(t) = a \cos \omega t + \frac{b - a \cos \omega T}{\sin \omega T} \sin \omega t$$

が上の境界条件を満たす調和振動子の微分方程式の唯一つの解になる.

$\sin \omega T = 0$ の場合. そのとき $\omega T = m\pi, m \in \mathbb{Z}_{>0}$ と書け, $\cos \omega T = (-1)^m$ となる. 上の境界条件を満たす解が存在するための必要十分条件は

$$b = (-1)^m a$$

となる. そのとき,

$$x(t) = a \cos \omega t + B \sin \omega t = x(0) \cos \omega t + B \sin \omega t$$

は上の境界条件を満たす調和振動子の微分方程式の解になる(B はなんでもよい).

2.4 調和振動子のNeumann型境界値問題

$T > 0$ と与えられた定数 a, b に対して, 調和振動子 $\ddot{x} = -\omega^2 x$ の解で境界条件

$$\dot{x}(0) = a\omega, \quad \dot{x}(T) = b\omega$$

を満たすものをすべて求めてみよう. これに一般解 $x(t) = A \cos \omega t + B \sin \omega t$ から得られる

$$\dot{x}(t) = \omega(-A \sin \omega t + B \cos \omega t)$$

を代入すると

$$B = a, \quad -A \sin \omega T + B \cos \omega T = b.$$

B は $B = a$ によって一意的に決まるが, A については $\sin \omega T$ が 0 であるか否かによって場合分けが必要になる.

$\sin \omega T \neq 0$ の場合. そのとき, 任意の a, b の組み合わせに対して, A, B が次のように一意的に決まる:

$$B = a, \quad A = \frac{a \cos \omega T - b}{\sin \omega T}.$$

すなわち,

$$x(t) = \frac{a \cos \omega T - b}{\sin \omega T} \cos \omega t + a \sin \omega t$$

が上の境界条件を満たす調和振動子の微分方程式の唯一つの解になる.

$\sin \omega T = 0$ の場合. そのとき $\omega T = m\pi, m \in \mathbb{Z}_{>0}$ と書け, $\cos \omega T = (-1)^m$ となる. 上の境界条件を満たす解が存在するための必要十分条件は

$$b = (-1)^m a$$

となる. そのとき,

$$x(t) = A \cos \omega t + a \sin \omega t = A \cos \omega t + \dot{x}(0) \sin \omega t$$

は上の境界条件を満たす調和振動子の微分方程式の解になる(A はなんでもよい).

3 BVProblemの使い方

https://docs.juliadiffeq.org/latest/tutorials/bvp_example.html (https://docs.juliadiffeq.org/latest/tutorials/bvp_example.html)

単振子:

$$\ddot{\theta} = -\frac{g}{\ell} \sin \theta.$$

```

1 ## θ = u[1]
2 ## dθ/dt = u[2]
3
4 const g = 9.81
5 const ℓ = 1.0
6 tspan = (0.0,pi/2)
7 function simplependulum!(du,u,p,t)
8     θ = u[1]
9     dθ = u[2]
10    du[1] = dθ
11    du[2] = -(g/ℓ)*sin(θ)
12 end

```

Out[2]: simplependulum! (generic function with 1 method)

3.1 境界条件1

[0, T] 上で

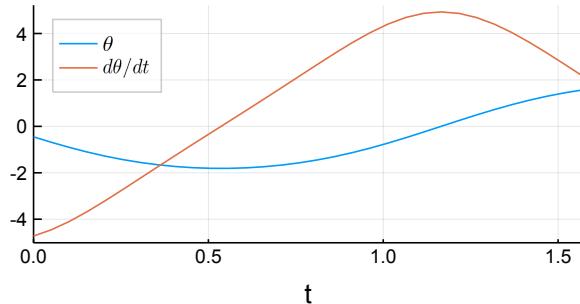
$$\theta(T/2) = -\frac{\pi}{2}, \quad \theta(0) = \frac{\pi}{2}.$$

```

1 function bc1!(residual, u, p, t)
2     residual[1] = u[end÷2][1] + pi/2 # the solution at the middle of the time span should be -pi/2
3     residual[2] = u[end][1] - pi/2 # the solution at the end of the time span should be pi/2
4 end
5 bvp1 = BVPProblem(simplependulum!, bc1!, [pi/2,pi/2], tspan)
6 sol1 = solve(bvp1, GeneralMIRK4(), dt=0.05)
7 plot(sol1, label=["\$\\theta\$", "\$d\\theta/dt\$"], lw=1, legend=:topleft, size=(400, 200))

```

Out[3]:



3.2 境界条件1 (別の方法)

[0, T] 上で

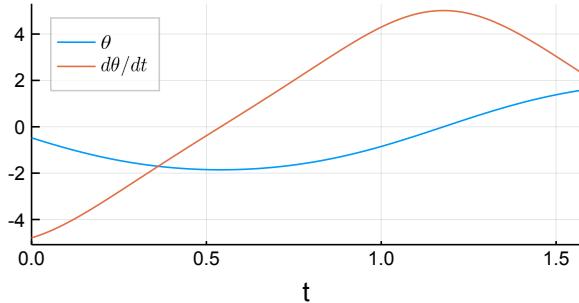
$$\theta(T/2) = -\frac{\pi}{2}, \quad \theta(0) = \frac{\pi}{2}.$$

```

In [4]: 1 u₀_2 = [-1.6, -1.7] # the initial guess
2 ▼ function bc3!(residual, sol, p, t)
3     residual[1] = sol(pi/4)[1] + pi/2 # use the interpolation here, since indexing will be wrong for adaptive
4     methods
5     residual[2] = sol(pi/2)[1] - pi/2
6 end
7 bvp3 = BVPProblem(simplependulum!, bc3!, u₀_2, tspan)
8 sol3 = solve(bvp3, Shooting(Vern7()))
9 plot(sol3, label=["$\theta$","$d\theta/dt$"], lw=1, legend=:topleft, size=(400, 200))

```

Out[4]:



3.3 境界条件2

$[0, T]$ 上で

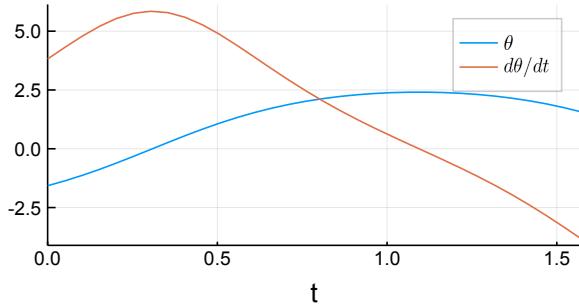
$$\theta(0) = -\frac{\pi}{2}, \quad \theta(T) = \frac{\pi}{2}$$

```

In [5]: 1 ▼ function bc2!(residual, u, p, t) # u[1] is the beginning of the time span, and u[end] is the ending
2     residual[1] = u[1][1] + pi/2 # the solution at the beginning of the time span should be -pi/2
3     residual[2] = u[end][1] - pi/2 # the solution at the end of the time span should be pi/2
4 end
5 bvp2 = TwoPointBVProblem(simplependulum!, bc2!, [pi/2,pi/2], tspan)
6 sol2 = solve(bvp2, MIRK4(), dt=0.05) # we need to use the MIRK4 solver for TwoPointBVProblem
7 plot(sol2, label=["$\theta$","$d\theta/dt$"], lw=1, legend=:topright, size=(400, 200))

```

Out[5]:



4 調和振動子の数値解

調和振動子

$$\ddot{x} = -\omega^2 x.$$

を数値的に解いてみよう。

```

In [6]: 1 ▼ function harmonicoscillator!(du, u, p, t)
2     ω = p[1]
3     du[1] = u[2]
4     du[2] = - ω^2*u[1]
5 end
6
7 ▼ function plot_harmonicoscillator(; ω=5.0, a=0.0, b=1.0, T=2π)
8     prob = ODEProblem(harmonicoscillator!, [a,b], (0,T), [ω])
9     sol = solve(prob, dt=0.005)
10    L = round(Int, 8π/T)
11    kpil(k) = isone(L) ? "\$\$($k)\$\pi\$" : "\$\$\\frac{($k)}{\pi}\$\$"
12    xticks_num = 0:π:L
13    xticks_str = [
14        "\$0\$\"
15        "\$\$($kpil(1))\$\"
16        "\$\$($kpil(k))\$\$" for k in 2:length(xticks_num)-1];
17    ]
18    P1 = plot(title="\$x(t):\$\$; \omega = \$ω\$\$", titlefontsize=12, legend=false)
19    plot!(xticks = (xticks_num, xticks_str))
20    plot!(sol, vars=(0,1), label="\$x\$\$", xlabel=" ", size=(400, 165), lw=1, color=:blue)
21    P2 = plot(title="\$dx(t)/dt:\$\$; \omega = \$ω\$\$", titlefontsize=12, legend=false)
22    plot!(xticks = (xticks_num, xticks_str))
23    plot!(sol, vars=(0,2), label="\$dx/dt\$\$", xlabel=" ", size=(400, 160), lw=1, color=:red)
24    plot(P1, P2, size=(700, 160))
25 end

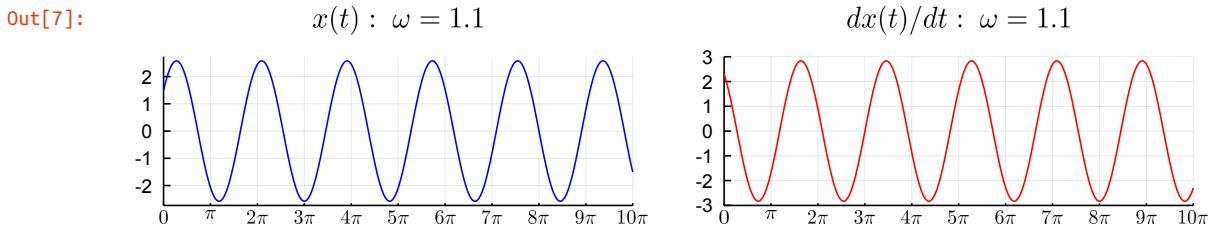
```

Out[6]: plot_harmonicoscillator (generic function with 1 method)

4.1 初期値問題

$$x(0) = a, \quad \dot{x}(0) = b.$$

In [7]: 1 plot_harmonicoscillator(ω=1.1, a=1.5, b=2.3, T=10π)



4.2 齊次Dirichlet境界条件

$T = 2\pi$ とし,

$$x(0) = x(T) = 0$$

と区間 $[0, T]$ の両端で 0 になる微分方程式

$$\ddot{x} = -\omega^2 x$$

の解を探してみる. $\omega = n/2, n = 1, 2, 3, \dots$ のとき,

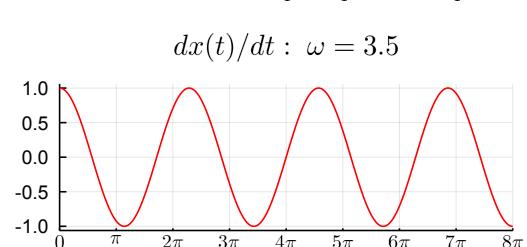
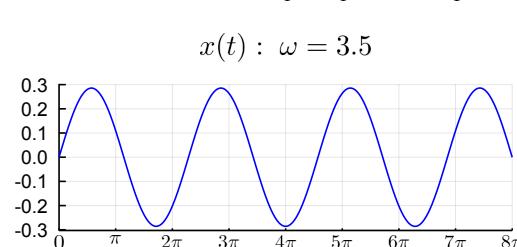
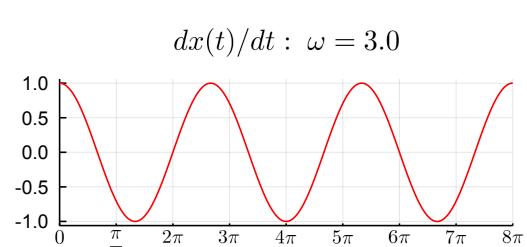
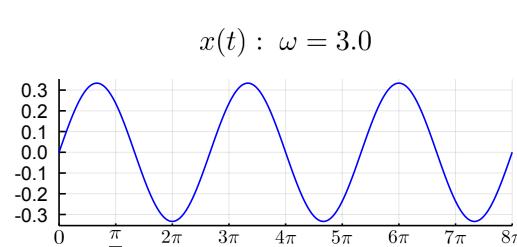
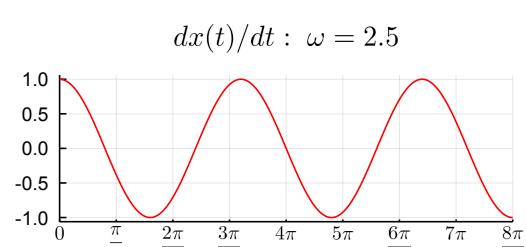
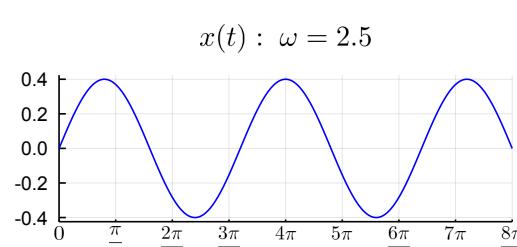
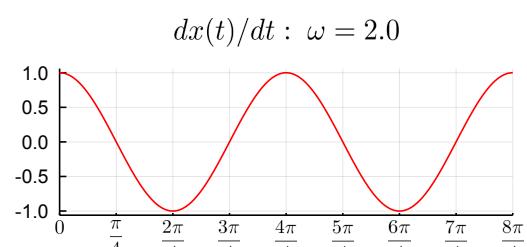
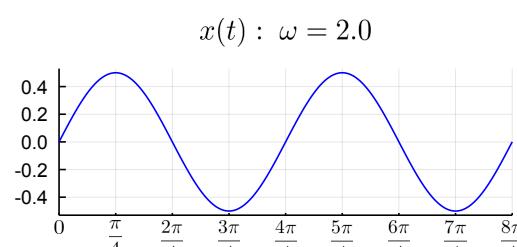
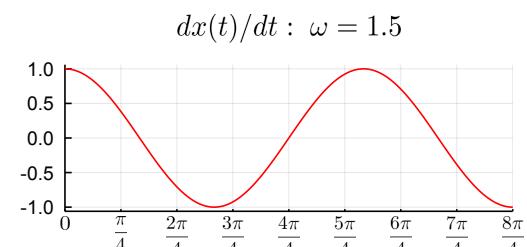
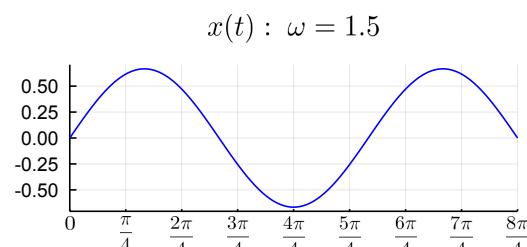
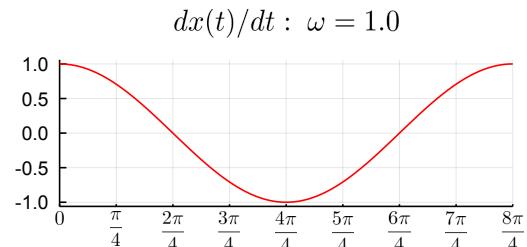
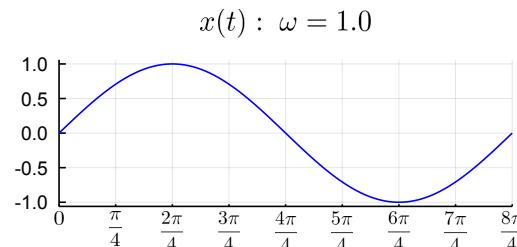
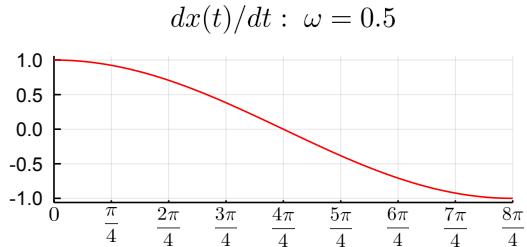
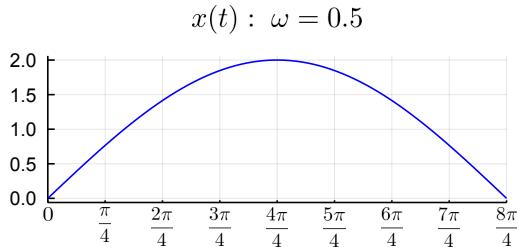
$$x(t) = \sin \frac{nt}{2}$$

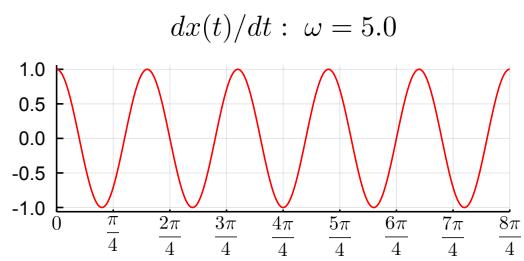
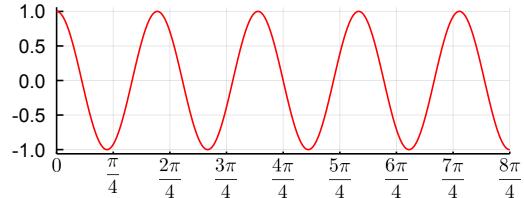
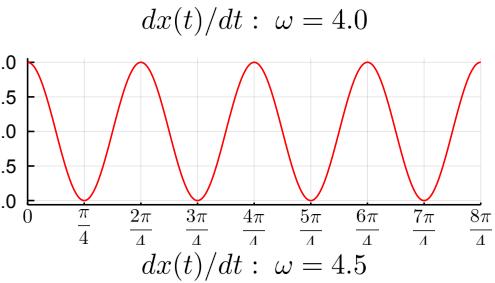
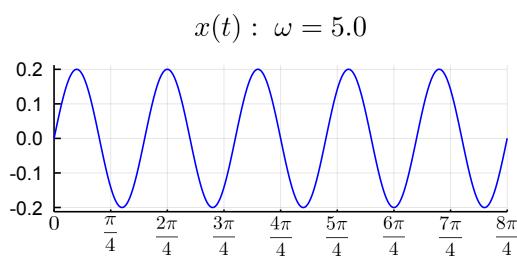
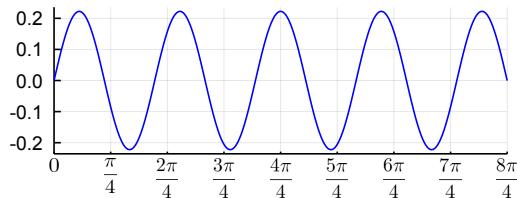
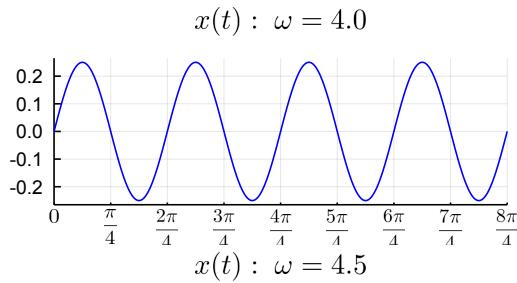
は上の条件を満たす解になっている.

```

1 ▼ for k in 1:10
2     plot_harmonicoscillator(; ω=0.5k, a=0.0, b=1.0, T=2π) ▶ display
3 end

```





4.3 齊次Neumann境界条件

$T = 2\pi$ とし,

$$\dot{x}(0) = \dot{x}(T) = 0$$

と区間 $[0, T]$ の両端で微分が 0 になる微分方程式

$$\ddot{x} = -\omega^2 x$$

の解を探してみる. ω が 2 分の整数のとき

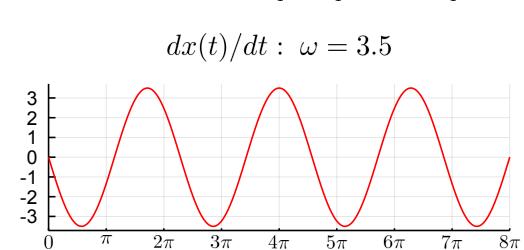
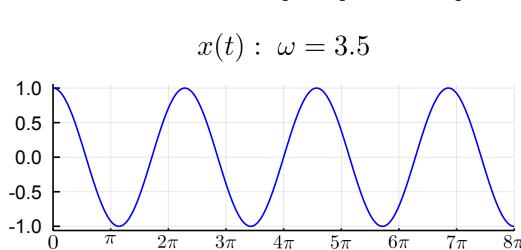
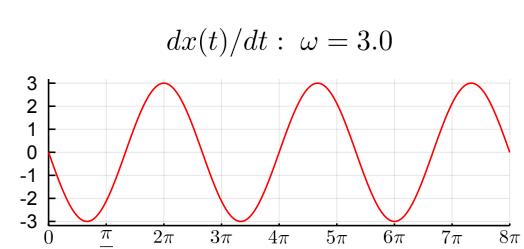
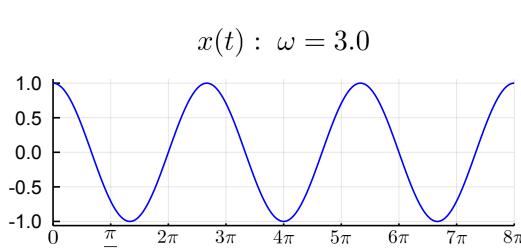
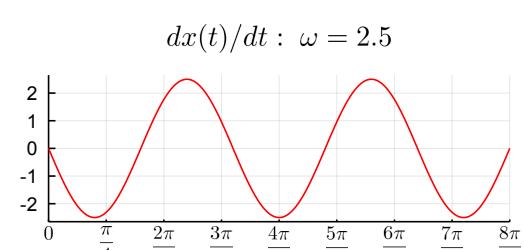
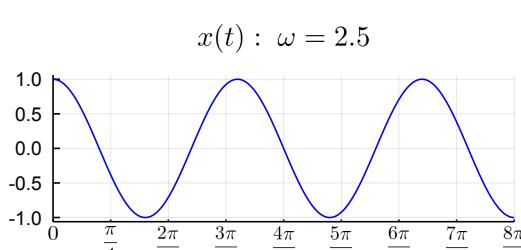
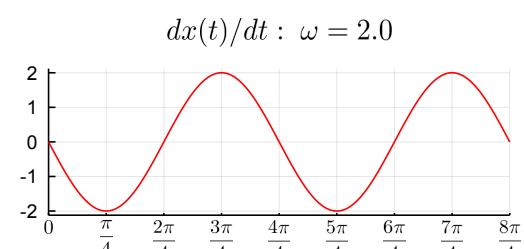
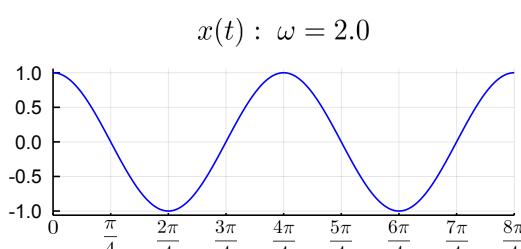
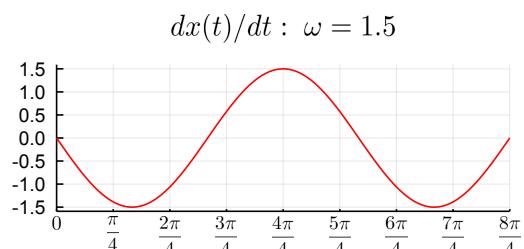
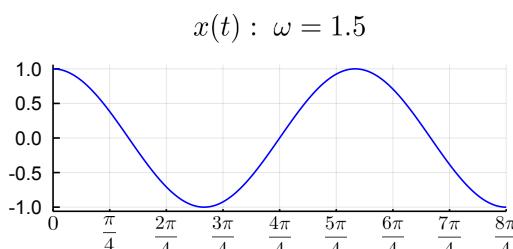
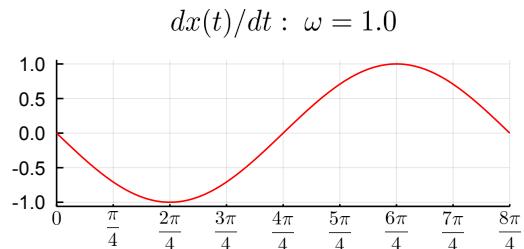
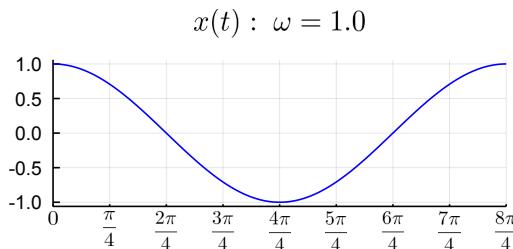
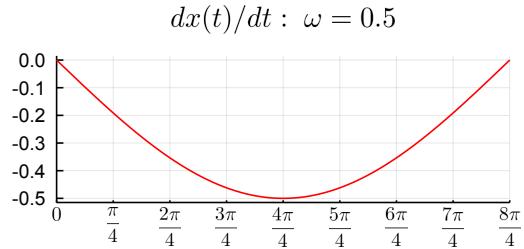
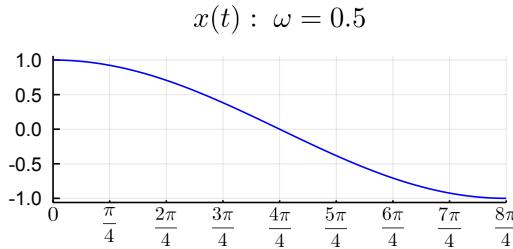
$$x(t) = \cos(\omega t)$$

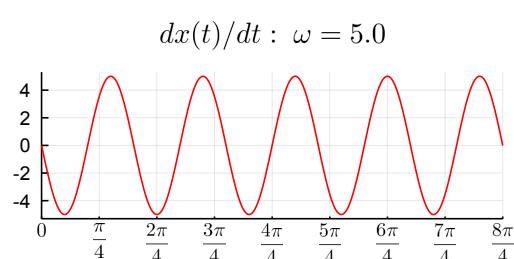
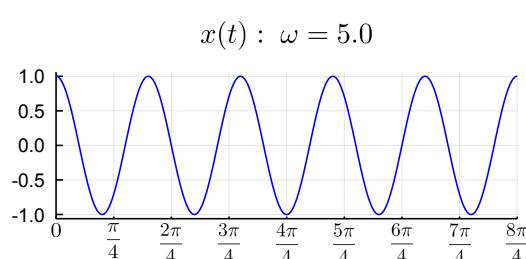
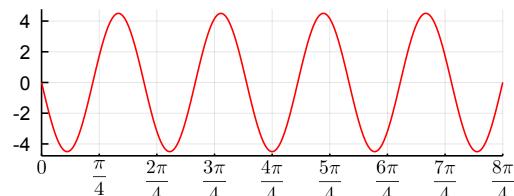
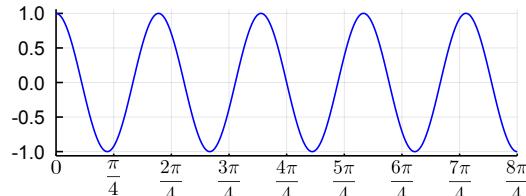
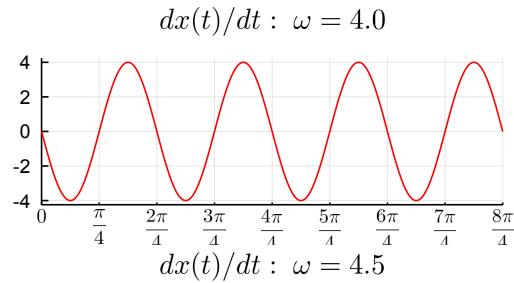
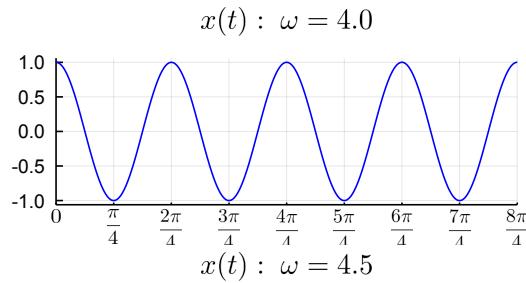
は上の条件を満たす解になっている.

```

1 ▼ for k in 1:10
2     plot_harmonicoscillator(; ω=0.5k, a=1.0, b=0.0, T=2π) ▶ display
3 end

```





4.4 非齊次Dirichlet境界条件

$\omega > 0, T > 0, \sin \omega T \neq 0$ のとき,

$$\ddot{x} = -\omega^2 x, \quad x(0) = a, \quad x(T) = b$$

の解は唯一つ存在し、次のように表わされる:

$$x(t) = a \cos \omega t + \frac{b - a \cos \omega T}{\sin \omega T} \sin \omega t$$

In [10]:

```
1 ▼ ##### parameters
2
3 ω = 4/3
4 T = 2π
5 a, b = 1.0, 2.0;
```

In [11]:

```
1 ▼ ##### exact solution
2
3 fD(ω, T, a, b, t) = a*cos(ω*t) + (b - a*cos(ω*T))/sin(ω*T)*sin(ω*t)
4 dfD(ω, T, a, b, t) = -a*ω*sin(ω*t) + (b - a*cos(ω*T))/sin(ω*T)*ω*cos(ω*t)
```

Out[11]: dfD (generic function with 1 method)

In [12]:

```
1 ▼ ##### numerical solution
2
3 ▼ function bcD!(residual, u, p, t)
4     residual[1] = u[1][1] - p[2]
5     residual[2] = u[end][1] - p[3]
6 end
7 bvpD = BVProblem(harmonicoscillator!, bcD!, [a,0], (0,T), [ω, a, b])
8 @time sold = solve(bvpD, GeneralMIRK4(), dt=0.01);
```

18.319562 seconds (305.94 M allocations: 23.267 GiB, 18.69% gc time)

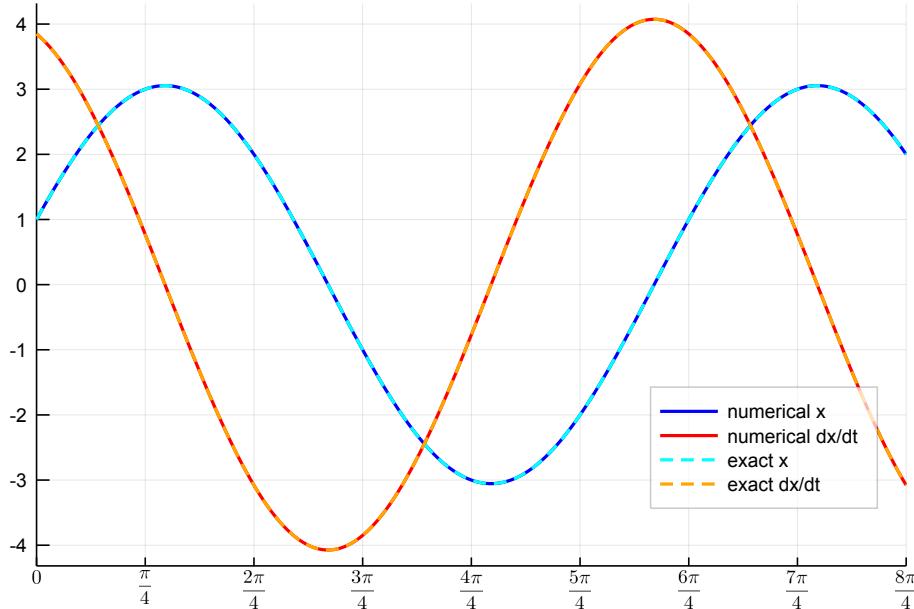
In [13]:

```

1 L = round(Int, 8π/T)
2 kpiL(k) = isone(L) ? "\$(k)\\"pi\$" : "\$\\frac{\$(k)\\"pi}{\$L}\$"
3 xticks_num = 0:π/L:T
4 ▾ xticks_str = [
5   "\$0\$"
6   "\$(kpiL(\""))\$"
7   ["\$(kpiL(k))\$" for k in 2:length(xticks_num)-1];
8 ]
9 xticks = (xticks_num, xticks_str)
10
11 plot(legend=:bottomright, xticks=xticks, yticks=-4:1:4)
12 plot!(sold, vars=(0,1), xlabel=" ", label="numerical x", lw=2, color=:blue)
13 plot!(sold, vars=(0,2), xlabel=" ", label="numerical dx/dt", lw=2, color=:red)
14 ts = range(0, T, length=200)
15 plot!(ts, fD.(ω, T, a, b, ts), ls=:dash, lw=2, color=:cyan, label="exact x")
16 plot!(ts, dfD.(ω, T, a, b, ts), ls=:dash, lw=2, color=:orange, label="exact dx/dt")

```

Out[13]:



4.5 非齊次Neumann境界条件

$\omega > 0, T > 0, \sin \omega T \neq 0$ のとき,

$$\ddot{x} = -\omega^2 x, \quad \dot{x}(0) = a\omega, \quad \dot{x}(T) = b\omega$$

の解は唯一つ存在し、次のように表わされる:

$$x(t) = \frac{a \cos \omega T - b}{\sin \omega T} \cos \omega t + a \sin \omega t$$

In [14]:

```

1 ▾ ### parameters
2
3 T = 2π
4 ω = 4/3
5 a, b = 1.0/ω, 2.0/ω

```

Out[14]: (0.75, 1.5)

In [15]:

```

1 ▾ ### exact solution
2
3 fN(ω, T, a, b, t) = (a*cos(ω*T)-b)/sin(ω*T)*cos(ω*t) + a*sin(ω*t)
4 dfN(ω, T, a, b, t) = -(a*cos(ω*T)-b)/sin(ω*T)*ω*sin(ω*t) + a*ω*cos(ω*t)

```

Out[15]: dfN (generic function with 1 method)

In [16]:

```

1 ## numerical solution
2
3 function bcN!(residual, u, p, t)
4     residual[1] = u[1][2] - p[2]*p[1]
5     residual[2] = u[end][2] - p[3]*p[1]
6 end
7 bvpN = BVProblem(harmonicoscillator!, bcN!, [0,a], (0,T), [ω,a,b])
8 @time soln = solve(bvpN, GeneralMIRK4(), dt=0.01);

```

17.577171 seconds (305.38 M allocations: 23.238 GiB, 19.10% gc time)

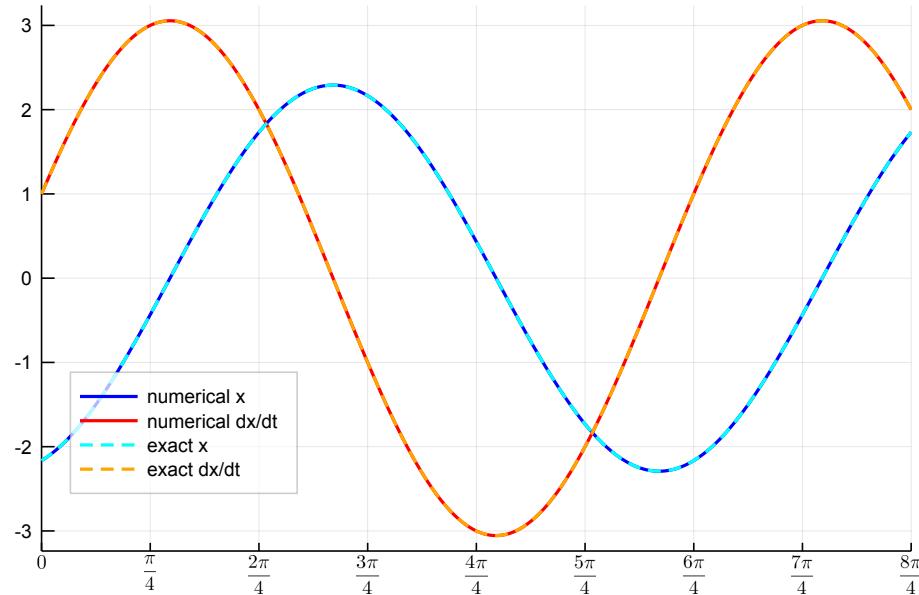
In [66]:

```

1 L = round(Int, 8π/T)
2 kpiL(k) = isone(L) ? "\$\$(k)\\"pi\$" : "\$\\frac{\$(k)}{\pi}\$"
3 xticks_num = 0:π/L:T
4 xticks_str = [
5     "\$0\$\"
6     "\$\$(kpiL(""))\$\"
7     ["\$\$(kpiL(k))\$\" for k in 2:length(xticks_num)-1];
8 ]
9 xticks = (xticks_num, xticks_str)
10
11 plot(legend=:bottomleft, xticks=xticks, yticks=-4:1:4)
12 plot!(soln, vars=(0,1), xlabel=" ", label="numerical x", lw=2, color=:blue)
13 plot!(soln, vars=(0,2), xlabel=" ", label="numerical dx/dt", lw=2, color=:red)
14 ts = range(0, T, length=200)
15 plot!(ts, fN.(ω, T, a, b, ts), ls=:dash, lw=2, color=:cyan, label="exact x")
16 plot!(ts, dfN.(ω, T, a, b, ts), ls=:dash, lw=2, color=:orange, label="exact dx/dt")

```

Out[66]:



In []:

1

定数係数線形常微分方程式(齊次)

- Author: 黒木玄
- Date: 2019-04-23
- Repository: <https://github.com/genkuroki/DifferentialEquations> (<https://github.com/genkuroki/DifferentialEquations>)

このファイルは [nbviewer \(https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/01-1%20Van%20der%20Pol%20oscillator.ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/01-1%20Van%20der%20Pol%20oscillator.ipynb) でも閲覧できる。

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) と [Jupyter環境 \(https://jupyter.org/\)](https://jupyter.org/) の簡単な解説については次を参照せよ:

- [JuliaとJupyterのすすめ \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/msfd28genkuroki.ipynb?flush_cached=true)

[Julia言語 \(https://julialang.org/\)](https://julialang.org/) 環境の整備の仕方については次を参照せよ:

- [Julia v1.1.0 の Windows 8.1 へのインストール \(https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb\)](https://nbviewer.jupyter.org/github/genkuroki/msfd28/blob/master/install.ipynb)

目次

1 2階単独の定数係数線形常微分方程式

- 1.1 2階単独の定数係数線形常微分方程式と定数係数で線形な3項間漸化式の類似性
- 1.2 方程式を解くときの考え方
- 1.3 2階単独の定数係数線形常微分方程式の解全体が2次元のベクトル空間になること
- 1.4 解の見付け方(1)
- 1.5 解の見付け方(2)
- 1.6 結果のまとめ
- 1.7 一次独立性の証明
 - 1.7.1 $\alpha \neq \beta$ のときの $e^{\alpha t}, e^{\beta t}$ の一次独立性の証明
 - 1.7.2 $e^{\alpha t}, te^{\alpha t}$ の一次独立性の証明

2 高階の場合

- 2.1 線形微分作用素
- 2.2 定数係数線形常微分方程式の微分作用素を用いた表示
- 2.3 重解がない場合の定数係数線形常微分方程式の解法
 - 2.3.1 定数係数でない場合の齊次1階の線形常微分方程式の解法
- 2.4 重解がある場合の定数係数線形常微分方程式の解法の準備
 - 2.4.1 定数係数でない場合の非齊次1階の線形常微分方程式の解法
- 2.5 重解がある場合の定数係数線形常微分方程式の解法

3 一次独立性の証明

- 3.1 Wronskian
 - 3.1.1 函数の組の一次独立性の十分条件
 - 3.1.2 線形常微分方程式の解のWronskian
 - 3.1.3 行列式の微分
- 3.2 Vandermondeの行列式
 - 3.2.1 Vandermondeの行列式が差積になることの別証明
- 3.3 互いに異なる α_j に対する $e^{\alpha_j x}$ 達の一次独立性
- 3.4 互いに異なる α_j に対する(多項式函数) $\times e^{\alpha_j x}$ 達の一次独立性
- 3.5 Vandermondeの行列式の公式の一般化
 - 3.5.1 互いに異なる α_j に対する(多項式函数) $\times e^{\alpha_j x}$ 達の一次独立性の別証明
 - 3.5.2 WolframAlphaによる一般化されたVandermondeの行列式の計算例
 - 3.5.3 SymPyによる一般化されたVandermondeの行列式の計算例

In [1]:

```

1  using Base64
2
3 ▼ showing(mime, fn; scale "") = open(fn) do f
4     base64 = base64encode(f)
5     if scale == ""
6         display("text/html", """""")
7     else
8         display("text/html", """""")
9     end
10    end
11
12  using SymPy: SymPy, sympy, Sym, @syms, @vars, oo

```

In [2]:

```
1 showing("image/jpeg", "images/generalized-Vandermonde-1.jpg", scale="70%")
```

factor det{{1,0,1,1,1},{a,1,c,d,f},{a^2,2a,c^2,d^2,f^2},{a^3,3a^2,c^3,d^3,f^3},{a^4,4a^3,c^4,d^4,f^4}}

Input interpretation

factor

$$\begin{vmatrix} 1 & 0 & 1 & 1 & 1 \\ a & 1 & c & d & f \\ a^2 & 2a & c^2 & d^2 & f^2 \\ a^3 & 3a^2 & c^3 & d^3 & f^3 \\ a^4 & 4a^3 & c^4 & d^4 & f^4 \end{vmatrix}$$

Result

$$-(a - c)^2 (a - d)^2 (a - f)^2 (c - d) (c - f) (d - f)$$

1 2階単独の定数係数線形常微分方程式

複素数値函数 $x = x(t)$ に関する次の線形常微分方程式の解をすべて求めたい:

$$\ddot{x} + p\dot{x} + qx = 0. \quad (*)$$

ここで p, q は複素定数である(応用上は p, q は実数になることが多い). これを**2階単独の定数係数線形常微分方程式**と呼ぶ.

1.1 2階単独の定数係数線形常微分方程式と定数係数で線形な3項間漸化式の類似性

方程式(*)は高校数学で習う次の定数係数で線形な3項間漸化式に似ている:

$$a_{n+2} + pa_{n+1} + qa_n = 0. \quad (\star)$$

これを満たす数列 $(a_n)_{n \in \mathbb{Z}} = (\dots, a_{-2}, a_{-1}, a_0, a_1, a_2, \dots)$ をこの漸化式の解と呼ぶ. 漸化式(\star)の解をすべて求める問題と方程式(*)の解をすべて求める問題は「ほぼ同じ問題である」と言ってよいほど似ている. 漸化式(\star)の解き方を復習し直せば, 方程式(*)に関する以下の解説の内容もすんなり理解できるものと思われる.

注意: 離散化された直線 \mathbb{Z} 上の3項間漸化式

$$-a_{n-1} + 2a_n - a_{n+1} = Ea_n$$

は直線 \mathbb{R} 上の頻出形の微分方程式

$$-\frac{d^2u}{dx^2} = \omega^2 u$$

の離散化とみなされるので特に重要である. その形の漸化式は $\alpha = 2 - E$ とおくと

$$a_{n-1} + a_{n+1} = \alpha a_n$$

と書き直される. この形の漸化式に関する非常に詳しい分析については

- [A02 Diagonalization of Cartan matrices of classical types](https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/A02%20Diagonalization%20of%20Cartan%20matrices.ipynb)
(<https://nbviewer.jupyter.org/github/genkuroki/DifferentialEquations/blob/master/A02%20Diagonalization%20of%20Cartan%20matrices.ipynb>)

を参照せよ. □

1.2 方程式を解くときの考え方

方程式は解く手続きは以下の2つに分けられる.

1. どのような手段でもよいので、とにかく、方程式の解をたくさん見付ける。
2. 見付けた解で方程式の解がすべて尽くされることを確認する。

ポイントは解を見付けるための手段は何であっても構わないということである。

とにかくどんな「汚い手」を使っていても、十分たくさんの解を見付けることができれば「勝ち」になる。

1.3 2階単独の定数係数線形常微分方程式の解全体が2次元のベクトル空間になること

このノートでは次の結果を認めて使うことにする。

- 2階単独の定数係数線形常微分方程式(*)の解全体の集合は2次元のベクトル空間になる。

線形代数の教科書や講義などでベクトル空間の公理的な定義を学んだが、その定義をピンとくるまで理解できなかった人は、「ベクトル空間になること」は「一次結合で閉じている(空でない)集合になっていること」とほぼ同義であると考えてよい。実用的にはそれで困らないと思われる。(注意: 「一次独立」の概念はそのように難な扱いが許されない。)

「ベクトル空間になること」は(認めて使わなくても)容易に確認できる。実際に確認しておこう。まず、定数函数 $f(t) = 0$ は方程式(*)の解になっていることがすぐにわかる。これで方程式(*)の解全体の集合が空集合でないことがわかった。函数 $f(t)$ と $g(t)$ が方程式

$$\ddot{x} + p\dot{x} + qx = 0 \quad (*)$$

の解であると仮定する。すなわち

$$f''(t) + pf'(t) + qf(t) = 0, \quad g''(t) + pg'(t) + qg(t) = 0$$

が成立していると仮定する。このとき、定数 a, b について、

$$\begin{aligned} & (af(t) + bg(t))'' + p(af(t) + bg(t))' + q(af(t) + bf(t)) \\ &= (af''(t) + bg''(t)) + p(af'(t) + bg'(t)) + q(af(t) + bf(t)) \\ &= a(f''(t) + pf'(t) + qf(t)) + b(g''(t) + pg'(t) + qg(t)) \\ &= 0. \end{aligned}$$

これで、方程式(*)の解全体の集合が一次結合で閉じていることがわかった。これで方程式(*)の解全体の集合がベクトル空間をなすことがわかったと考えてよい。

そのベクトル空間が2次元になることは以下で認めて使うことにする。

ベクトル空間が2次元になることの定義は、そのベクトル空間の2つの要素で一次独立なものが存在し、そのベクトル空間の任意の要素がその2つの一次独立なベクトルの一次結合で表せることである。(再注意: 一次独立性の概念は重要なので正確な意味と直観的な意味を必ず復習しておくこと!)

ゆえに、上の結果を認めると、方程式(*)のすべての解を求めるためには、一次独立な2つの解 $f(t), g(t)$ を見付ければ十分だということがわかる。そのとき、方程式(*)の任意の解 $x(t)$ はそれらの一次結合

$$x(t) = af(t) + bg(t)$$

の形で一意的に表わされる。我々はこのような $f(t), g(t)$ を何らかの方法で求めることができればよいということになった。

1.4 解の見付け方(1)

方程式

$$\ddot{x} + p\dot{x} + qx = 0 \quad (*)$$

の解で $x(t) = e^{\lambda t}$ の形のものを見つけてみよう。 $x = x(t) = e^{\lambda t}$ を(*)に代入すると、

$$\lambda^2 e^{\lambda t} + p\lambda e^{\lambda t} + qe^{\lambda t} = 0.$$

これと

$$\lambda^2 + p\lambda + q = 0.$$

は同値である。したがって、

$$\lambda^2 + p\lambda + q = (\lambda - \alpha)(\lambda - \beta)$$

のとき, $f(t) = e^{\alpha t}$ と $g(t) = e^{\beta t}$ は方程式(*)の解である.

もしも $\alpha \neq \beta$ ならばそれらは異なる解であり, 一次独立であることを示せる(後で証明する). ゆえに $\alpha \neq \beta$ のとき, 方程式(*)の任意の解は

$$x(t) = ae^{\alpha t} + be^{\beta t}$$

と一意に表わされる.

1.5 解の見付け方(2)

$\lambda^2 + p\lambda + q = (\lambda - \alpha)(\lambda - \beta)$ で $\alpha \neq \beta$ の場合には方程式(*)の2つの異なる解 $e^{\alpha t}, e^{\beta t}$ が得られて, それらが一次独立になるので, それですべての解を求める手続きは実質的に終了する.

それでは $\alpha = \beta$ (重解の場合)はどうすればよいのだろうか?

$\lambda^2 + p\lambda + q = (\lambda - \alpha)^2$ の場合には $f(t) = e^{\alpha t}$ だけではなく, $g(t) = te^{\alpha t}$ も(*)の解になることを以下のようにして示せる:

$$\begin{aligned} g(t) &= te^{\alpha t}, \\ g'(t) &= e^{\alpha t} + \alpha te^{\alpha t} = (\alpha t + 1)e^{\alpha t}, \\ g''(t) &= \alpha e^{\alpha t} + \alpha(1 + \alpha t)e^{\alpha t} = (\alpha^2 t + 2\alpha)e^{\alpha t} \end{aligned}$$

より,

$$\begin{aligned} g''(t) + pg'(t) + qg(t) &= (2\alpha + \alpha^2 t + p(1 + \alpha t) + qt)e^{\alpha t} \\ &= (\alpha^2 + p\alpha + q)t + 2\alpha + p)e^{\alpha t} \\ &= 0. \end{aligned}$$

最後の等号で $\lambda^2 + p\lambda + q = (\lambda - \alpha)^2 = \lambda^2 - 2\alpha\lambda + \alpha^2$ であることを使つた. $\alpha^2 + p\alpha + q = 0$ と $p = -2\alpha, q = \alpha^2$ が成立している.(以上の計算は実は不必要に煩雑になつておる, 高階の場合に一般化することが難しくなつてゐる. 後で高階の場合についても通用する楽な考え方を説明する.)

さらに, $f(t) = e^{\alpha t}, g(t) = te^{\alpha t}$ は一次独立なので(後で証明する), $\alpha = \beta$ のとき方程式(*)の任意の解は

$$x(t) = ae^{\alpha t} + bte^{\alpha t}$$

と一意に表わされる.

1.6 結果のまとめ

微分方程式

$$\ddot{x} + p\dot{x} + qx = 0 \quad (*)$$

の解は以下のようにして求められる.

$\lambda^2 + p\lambda + q = (\lambda - \alpha)(\lambda - \beta)$ であるとする.

$\alpha \neq \beta$ のとき, 方程式(*)の任意の解は次のように一意に表わされる:

$$x(t) = ae^{\alpha t} + be^{\beta t}.$$

$\alpha = \beta$ のとき, 方程式(*)の任意の解は次のように一意に表わされる:

$$x(t) = ae^{\alpha t} + bte^{\alpha t}.$$

1.7 一次独立性の証明

1.7.1 $\alpha \neq \beta$ のときの $e^{\alpha t}, e^{\beta t}$ の一次独立性の証明

$\alpha \neq \beta$ と仮定し, $a, b \in \mathbb{C}$ とし,

$$ae^{\alpha t} + be^{\beta t} = 0$$

であると仮定する. $e^{\alpha t}, e^{\beta t}$ の一次独立性を証明するためには $a = b = 0$ となることを示せばよい. 上の式の両辺を微分すると,

$$\alpha ae^{\alpha t} + \beta be^{\beta t} = 0.$$

さらに上の2つの公式において $t = 0$ とおくと,

$$a + b = 0, \quad \alpha a + \beta b = 0.$$

左の等式の両辺に β をかけたものから, 右の等式を引くと, $(\beta - \alpha)a = 0$ が得られる. $\alpha \neq \beta$ より, $\beta - \alpha \neq 0$ となるので $a = 0$ となる. ゆえに $a + b = 0$ より $b = 0$ も得られる. 以上によって $e^{\alpha t}, e^{\beta t}$ が一次独立であることが示された.

注意: $a + b = 0, \alpha a + \beta b = 0$ は行列を使えば

$$\begin{bmatrix} 1 & 1 \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

と書き直される. 左辺の行列の行列式は $\beta - \alpha$ なので $\alpha \neq \beta$ ならば 0 にならない. 一般に正方行列について, その行列式が 0 にならないこととその逆行列が存在することは同値である. ゆえに $\alpha \neq \beta$ のとき, 左辺の行列の逆行列を両辺にかけることによって,

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

を得ることができる. この筋道であれば高階の場合に一般化しやすい. 高階の場合への一般化では [Vandermondeの行列式](#) (<https://www.google.com/search?q=Vandermonde%E3%81%AE%E8%A1%8C%E5%88%97%E5%BC%8F>) が出て来る. \square

Vandermondeの行列式の様な重要な数学的対象がこのような形で自然に必要になる. 微分方程式論には数学のあらゆる分野の道具が総動員されることになる. だから, 微分方程式論を詳しく勉強すると, 自然に多くの数学について詳しくなることになる.もちろん, 線形代数についても詳しくなるであろう. 実際には線形代数のような基本的な数学的道具の価値は実際に応用される現場を見て初めて認識できることが多い.

1.7.2 $e^{\alpha t}, te^{\alpha t}$ の一次独立性の証明

$a, b \in \mathbb{C}$ とし,

$$ae^{\alpha t} + bte^{\alpha t} = 0$$

であると仮定する. $e^{\alpha t}, te^{\alpha t}$ の一次独立性を証明するためには $a = b = 0$ となることを示せばよい. 上の式の両辺を微分すると,

$$(\alpha a + b)e^{\alpha t} + \beta bte^{\alpha t} = 0.$$

さらに上の2つの公式において $t = 0$ とおくと,

$$a = 0, \quad \alpha a + b = 0.$$

これより, $a = b = 0$ となることがわかるので, $e^{\alpha t}, te^{\alpha t}$ が一次独立であることが示された.

注意: 横ベクトル値函数 $[e^{\alpha t}, te^{\alpha t}]$ の両辺を t で微分すると, $[e^{\alpha t} \alpha, e^{\alpha t} + te^{\alpha t}]$ になるが, これは行列を使って次のように表わされる:

$$\frac{d}{dt}[e^{\alpha t}, te^{\alpha t}] = [e^{\alpha t} \alpha, e^{\alpha t} + te^{\alpha t}] = [e^{\alpha t}, te^{\alpha t}] \begin{bmatrix} \alpha & 1 \\ 0 & \alpha \end{bmatrix}.$$

再右辺の 2×2 行列は基底 $e^{\alpha t}, te^{\alpha t}$ で張られる2次元の函数空間に作用する線形変換 d/dt の基底を用いた行列表示である. その行列表示は所謂 [Jordan標準形](#) (<https://www.google.com/search?q=Jordan%E6%A8%99%E6%BA%96%E5%BD%A2>) の形をしている. 高階の場合にも「重解を持つ場合」には同じようにして Jordan標準形の行列が現われる. \square

正方行列のJordan標準形の様な重要な数学的対象がこのような形で自然に必要になる. 微分方程式論には数学のあらゆる分野の道具が総動員されることになる. だから, 微分方程式論を詳しく勉強すると, 自然に多くの数学について詳しくなることになる. もちろん, 線形代数についても詳しくなるであろう. 実際には線形代数のような基本的な数学的道具の価値は実際に応用される現場を見て初めて認識できることが多い.

2 高階の場合

この節では $u = u(x)$ に関する次の微分方程式の解をすべて求める:

$$u^{(n)} + p_1 u^{(n-1)} + \cdots + p_{n-1} u' + p_n u = 0. \quad (*)$$

ここで $u^{(k)}$ は $u = u(x)$ の k 階の導函数を表しており, p_1, \dots, p_{n-1}, p_n は複素数の定数である. この微分方程式は n 階単独の定数係数線形常微分方程式と呼ばれる. (この節では気分を変えて, t ではなく, x の函数を扱う. 読者が記号に頼った悪しき思考に陥らないように, わざとこのようにすることにした.)

方程式(*)の解全体の空間が n 次元のベクトル空間になることを認めて使うことにする. ゆえに, 方程式(*)の n 個の解で一次独立になるものを見付けることができれば, 任意の解はそれらの一次結合で一意的に表わされることもわかる. したがって, 以下で方程式(*)の n 個の解で一次独立になるものを見付けることを目標とする.

2.1 線形微分作用素

函数 $f(x)$ をその導函数 $f'(x)$ に対応させる写像は $\frac{d}{dx}$ と書かれる。その写像は線形写像になっており、

$$\frac{d}{dx} f(x) = f'(x)$$

の左辺のように書かれる。これを一般化して、函数をその k 階の導函数に対応させる写像を

$$\left(\frac{d}{dx}\right)^k f(x) = f^{(k)}(x)$$

と書くことにする。さらに毎回 $\frac{d}{dx}$ と書くのは面倒なので、それを一文字の ∂ で表すことにする：

$$\partial = \frac{d}{dx}, \quad \partial^k = \left(\frac{d}{dx}\right)^k.$$

函数を函数に対応させる線形写像 P, Q の和と差と積とスカラー倍と函数 $a(x)$ 倍が

$$\begin{aligned} (P \pm Q)f(x) &= Pf(x) \pm Qf(x), \\ (PQ)f(x) &= P(Qf)(x), \\ (\alpha P)f(x) &= \alpha \cdot Pf(x), \\ (a(x)P)f(x) &= a(x) \cdot Pf(x) \end{aligned}$$

によって定義される。このようにして、函数を函数に対応させる線形写像(作用素、演算子、operatorなどと呼ばれる)達はまるで行列のように計算できる。(行列にも和と差と積とスカラー倍が定義されているのであった。) 行列の積が非可換であったのと同じように、特別な場合を除いて(後でその特別な場合が出て来る!)、 $PQ = QP$ のような計算を自由にできなくなることに注意せよ。

次の作用素は(線形常)微分作用素と呼ばれる：

$$P = a_n(x)\partial^n + \cdots + a_1(x)\partial + a_0(x).$$

この作用素は函数 $f(x)$ を次に対応させる線形写像になっている：

$$Pf(x) = a_n(x)f^{(n)}(x) + \cdots + a_1(x)f'(x) + a_0(x)f(x).$$

微分作用素は線形写像の重要な例になっている。

例: $\partial = d/dx$ と $a(x)$ をかける操作は非可換である：

$$\begin{aligned} a(x)\partial f(x) &= a(x)f(x), \\ \partial(a(x)f(x)) &= (a(x)f(x))' = a'(x)f(x) + a(x)f'(x) \end{aligned}$$

なので

$$(\partial a(x) - a(x)\partial)f(x) = \partial(a(x)f(x)) - a(x)\partial f(x) = a'(x)f(x).$$

以上より、作用素として、

$$\partial a(x) - a(x)\partial = a'(x)$$

となっていることがわかる。□

2.2 定数係数線形常微分方程式の微分作用素を用いた表示

定数係数の線形常微分作用素

$$\partial^n + p_1\partial^{n-1} + \cdots + p_{n-1}\partial + p_n$$

を使うと、微分方程式

$$u^{(n)} + p_1u^{(n-1)} + \cdots + p_{n-1}u' + p_nu = 0 \quad (*)$$

は

$$(\partial^n + p_1\partial^{n-1} + \cdots + p_{n-1}\partial + p_n)u = 0$$

と表わされる。

2.3 重解がない場合の定数係数線形常微分方程式の解法

定数倍と微分する操作は可換なので、定数係数線形常微分作用素はまるで ∂ の(可換な)多項式のように計算できる。ゆえに、

$$\lambda^n + p_1 \lambda^{n-1} + \cdots + p_{n-1} \lambda + p_n = (\lambda - \alpha_1) \cdots (\lambda - \alpha_n)$$

のとき、

$$\partial^n + p_1 \partial^{n-1} + \cdots + p_{n-1} \partial + p_n = (\partial - \alpha_1) \cdots (\partial - \alpha_n)$$

も成立している。このとき、方程式(*)は次のように書かれる：

$$(\partial - \alpha_1) \cdots (\partial - \alpha_n) u = 0.$$

$\partial - \alpha_j$ の積の順序は自由に交換できるので、もしも

$$(\partial - \alpha_j) u = u' - \alpha_j u = 0$$

すなわち

$$u' = \alpha_j u$$

が成立しているならば、 $(\partial - \alpha_1) \cdots (\partial - \alpha_n) u = 0$ も成立する。そして、

$$f_j(x) = e^{\alpha_j x}$$

は微分方程式 $u' = \alpha_j u$ の解なので、もしも n 個の α_j 達が互いに異なるならば、方程式(*)の n 個の異なる解が得られたことになる。実はそれらは一次独立になる(後で証明する)。

ゆえに、 n 個の α_j が互いに異なるとき、方程式(*)の任意の解は次のように一意に表わされる：

$$u(x) = a_1 e^{\alpha_1 x} + \cdots + a_n e^{\alpha_n x}.$$

注意: 以上の議論を見れば、 $e^{\alpha x}$ 型の函数が定数係数線形常微分方程式の解として出て来る理由がわかる。そのような解が出て来る理由は定数係数線形常微分方程式を与える定数係数線形常微分作用素が $(\partial - \alpha_1) \cdots (\partial - \alpha_n)$ と表され、その互いに可換な因子である $\partial - \alpha_j$ 達を作用させて 0 になる函数として $e^{\alpha_j x}$ が取れるからである。□

2.3.1 定数係数でない場合の齊次1階の線形常微分方程式の解法

一般に微分方程式 $u'(x) = a(x)u(x)$ の任意の解は

$$u(x) = C e^{\int a(x) dx}$$

と表わされる。 C は任意定数である。これもよく使われる。

例：作用素として $\partial x = x\partial + 1$ が成立することに注意すれば

$$(\partial - x)(\partial + x) = \partial^2 - x\partial + \partial x - x^2 = \partial^2 - x^2 + 1$$

が成立することがわかる。ゆえに、微分方程式

$$(\partial + x)u = 0, \quad \text{i.e.} \quad u' = -xu$$

の解

$$u(x) = e^{\int (-x) dx} = e^{-x^2/2}$$

は微分方程式

$$(\partial^2 - x^2 + 1)u = (\partial - x)(\partial + x)u = 0$$

の解になっていることがわかる。 $e^{-x^2/2}$ は量子調和振動子 (<https://www.google.com/search?q=%E9%87%8F%E5%AD%90%E8%AA%BF%E5%92%8C%E6%8C%AF%E5%8B%95%E5%AD%90>) の基底状態であり、0 番目の Hermite の多項式 (<https://www.google.com/search?q=Hermite%E3%81%AE%E5%A4%9A%E9%A0%85%E5%BC%8F>) の $e^{-x^2/2}$ 倍になっている。□

このように定数係数の場合に限らず、微分作用素の「因数分解」は数学的に重要なテクニックの一つになっている。

2.4 重解がある場合の定数係数線形常微分方程式の解法の準備

それでは n 個の α_j 達に重複がある場合にはどのようにして n 個の一次独立な解を見付けたらよいのだろうか？

そのためには作用素として,

$$e^{\alpha x} \partial e^{-\alpha x} = \partial - \alpha$$

となっていることが役に立つ. この公式は次のようにして示される:

$$\begin{aligned} e^{\alpha x} \partial(e^{-\alpha x} f(x)) &= e^{\alpha x} (e^{-\alpha x} f'(x) - \alpha e^{-\alpha x} f(x)) \\ &= f'(x) - \alpha f(x) \\ &= (\partial - \alpha) f(x). \end{aligned}$$

2.4.1 定数係数でない場合の非齊次1階の線形常微分方程式の解法

一般に作用素として,

$$e^{\int a(x) dx} \partial e^{-\int a(x) dx} = \partial - a(x)$$

が成立している. 実際,

$$\begin{aligned} e^{\int a(x) dx} \partial(e^{-\int a(x) dx} f(x)) &= e^{-\int a(x) dx} (e^{-\int a(x) dx} f'(x) - a(x) e^{-\int a(x) dx} f(x)) \\ &= f'(x) - a(x) f(x) \\ &= (\partial - a(x)) f(x). \end{aligned}$$

これもよく使われる.

例: 微分方程式 $(\partial - a(x))u = f(x)$ の解

$$u(x) = e^{\int a(x) dx} \int e^{-\int a(x) dx} f(x) dx$$

は以下のようにして得られる. $\partial - a(x) = e^{\int a(x) dx} \partial e^{-\int a(x) dx}$ なので, $(\partial - a(x))u = f(x)$ は

$$\partial(e^{-\int a(x) dx} u(x)) = e^{-\int a(x) dx} f(x)$$

に書き直される. 両辺を x で不定積分すれば,

$$e^{-\int a(x) dx} u(x) = \int e^{-\int a(x) dx} f(x) dx$$

これより, 上の解が得られる. \square

注意: 形式的には $\partial^{-1} = \int dx$ (微分の逆は積分)と書けるので,

$$(\partial - \alpha)^{-1} = (e^{\int a(x) dx} \partial e^{-\int a(x) dx})^{-1} = e^{\int a(x) dx} \int dx e^{-\int a(x) dx}$$

を $(\partial - \alpha)u(x) = f(x)$ の両辺に作用させて, 解 $u(x)$ が得られたと考えてもよい. \square

2.5 重解がある場合の定数係数線形常微分方程式の解法

一般に, 中間に挟まった $P^{-1} P$ がことごとく消えて,

$$(PAP^{-1})^m = PAP^{-1}PAP^{-1} \cdots PAP^{-1}PAP^{-1} = PA^m P^{-1}$$

となるので, 作用素としての公式 $e^{\alpha x} \partial e^{-\alpha x} = \partial - \alpha$ より,

$$(\partial - \alpha)^m = e^{\alpha x} \partial^m e^{-\alpha x}$$

となる. ゆえに方程式

$$(\partial - \alpha)^m u = 0$$

は

$$e^{\alpha x} \partial^m (e^{-\alpha x} u(x)) = 0$$

に書き直される. これは $e^{-\alpha x} u(x)$ を m 回微分すると消えることを意味しており, そうなることは, $e^{-\alpha x} u(x)$ が x に関する $m-1$ 次以下の多項式函数になることと同値である. ゆえに, 解 $u(x)$ は

$$u(x) = (a_{m-1} x^{m-1} + \cdots + a_1 x + a_0) e^{\alpha x}$$

と一意に表わされる。ゆえに方程式 $(\partial - \alpha)^m u = 0$ の解全体のなすベクトル空間の基底として、

$$e^{\alpha x}, xe^{\alpha x}, \dots, x^{m-1} e^{\alpha x}$$

が取れる。

以下では

$$\partial^n + p_1 \partial^{n-1} + \dots + p_{n-1} \partial + p_n = (\partial - \alpha_1)^{m_1} \cdots (\partial - \alpha_r)^{m_r}$$

で α_j 達が互いに異なると仮定する。

上で注意したことより、微分方程式

$$(\partial^n + p_1 \partial^{n-1} + \dots + p_{n-1} \partial + p_n)u = (\partial - \alpha_1)^{m_1} \cdots (\partial - \alpha_r)^{m_r} u = 0$$

の解として、以下が取れることはわかる：

$$e^{\alpha_j x}, xe^{\alpha_j x}, \dots, x^{m_j-1} e^{\alpha_j x}, \quad j = 1, 2, \dots, r.$$

実はこれらが上の微分方程式の解全体のなすベクトル空間の基底になっていることを示せる(一次独立性を後で示す)。

3 一次独立性の証明

この節では、函数達 $f_1(x), \dots, f_n(x)$ の一次独立性の証明を扱う。

3.1 Wronskian

函数 $f_1(x), \dots, f_n(x)$ に対する

$$W(f_1, \dots, f_n)(x) = \begin{vmatrix} f_1(x) & f_2(x) & \cdots & f_n(x) \\ f'_1(x) & f'_2(x) & & f'_n(x) \\ \vdots & \vdots & \cdots & \vdots \\ f_1^{(n-1)} & f_2^{(n-1)}(x) & \cdots & f_n^{(n-1)}(x) \end{vmatrix}$$

を f_1, \dots, f_n の **Wronskian** (ロンスキアン、ロンスキー行列式) と呼ぶ。

3.1.1 函数の組の一次独立性の十分条件

もしも、ある x_0 で $W(f_1, \dots, f_n)(x_0) \neq 0$ となっていれば f_1, \dots, f_n が一次独立である。

証明: $W(f_1, \dots, f_n)(x_0) \neq 0$, $a_1 f_1 + \dots + a_n f_n = 0$ と仮定する。 $a_1 = \dots = a_n = 0$ を示せばよい。 $a_1 f_1 + \dots + a_n f_n = 0$ の両辺を何度も微分することによって $a_1 f_1^{(k)} + \dots + a_n f_n^{(k)} = 0$ を得る。ゆえに

$$\begin{bmatrix} f_1(x) & f_2(x) & \cdots & f_n(x) \\ f'_1(x) & f'_2(x) & & f'_n(x) \\ \vdots & \vdots & \cdots & \vdots \\ f_1^{(n-1)} & f_2^{(n-1)}(x) & \cdots & f_n^{(n-1)}(x) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

を得る。左辺の中の行列の行列式は $x = x_0$ で 0 ないので、その逆行列が存在する。その逆行列を両辺に左からかけば $a_1 = \dots = a_n = 0$ が得られる。□

注意: f_1, \dots, f_n が一次独立であっても、ある a で $W(f_1, \dots, f_n)(a) = 0$ となるものが存在する場合がある。例えば、 $\varphi(x)$ を $x \leq -2$ で 0 になり、 $x \geq -1$ で 1 となる C^∞ 函数とすると、 $W(f_1, \dots, f_n)(0) \neq 0$ となる f_1, \dots, f_n について、 $g_j = f_j \varphi$ とおくと、 $W(g_1, \dots, g_n)(0) = W(f_1, \dots, f_n)(0) \neq 0$, $W(g_1, \dots, g_n)(-3) \neq 0$ となる。さらに強力な次のような例も存在する。□

例: $x \in \mathbb{R}$ について、 $f_1(x) = x^2$, $f_2(x) = |x| x$ のとき、 $f'_1(x) = 2x$, $f'_2(x) = 2|x|$ なので、

$$W(f_1, f_2)(x) = \det \begin{bmatrix} x^2 & |x| x \\ 2x & 2|x| \end{bmatrix} = x^2 \cdot 2|x| - 2x \cdot |x| x = 0.$$

このように f_1, f_2 のWronskianはすべての $x \in \mathbb{R}$ について 0 になる。一方、 $a_1 f_2 + a_2 f_2 = 0$ のとき、 $x = \pm 1$ を代入すると、

$$a_1 + a_2 = 0, \quad a_1 - a_2 = 0$$

となり、これから $a_1 = a_2 = 0$ が得られるので、 f_1, f_2 は一次独立であることがわかる。□

3.1.2 線形常微分方程式の解のWronskian

$f_1(x), \dots, f_n(x)$ は微分方程式

$$u^{(n)}(x) = a_1(x)u^{(n-1)}(x) + a_2(x)u^{(n-2)}(x) + \dots + a_n(x)u(x)$$

の解であると仮定する。このとき, $f_j^{(i-1)}$ を (i, j) 成分とする $n \times n$ 行列の第 i 行を $f^{(i-1)}$ と書くと,

$$\frac{df^{(n-1)}(x)}{dx} = a_1(x)f^{(n-1)}(x) + a_2(x)f^{(n-2)}(x) + \dots + a_n(x)f^{(0)}(x)$$

なので

$$\begin{aligned} \frac{d}{dx} W(f_1, \dots, f_n) &= \sum_{i=1}^{n-1} \det \begin{bmatrix} f^{(0)} \\ \vdots \\ df^{(i-1)}/dx \\ f^{(i)} \\ \vdots \\ f^{(n-1)} \end{bmatrix} + \det \begin{bmatrix} f^{(0)} \\ f^{(1)} \\ \vdots \\ f^{(n-2)} \\ df^{(n-1)}/dx \end{bmatrix} \\ &= \det \begin{bmatrix} f^{(0)} \\ f^{(1)} \\ \vdots \\ f^{(n-2)} \\ a_1(x)f^{(n-1)} + a_2(x)f^{(n-2)} + \dots + a_n(x)f^{(0)} \end{bmatrix} \\ &= \det \begin{bmatrix} f^{(0)} \\ f^{(1)} \\ \vdots \\ f^{(n-2)} \\ a_1(x)f^{(n-1)} \end{bmatrix} = a_1(x)W(f_1, \dots, f_n). \end{aligned}$$

3つ目の等号で, 行列式の中で第 $i = 1, \dots, n-1$ 行 $f^{(i-1)}$ の $a_{n+1-i}(x)$ 倍を第 n 行から引いた。ゆえに

$$W(f_1, \dots, f_n)(x) = \exp\left(\int_a^x a_1(\xi) d\xi\right) W(f_1, \dots, f_n)(a).$$

特に, $W(f_1, \dots, f_n)(a) \neq 0$ ならば $W(f_1, \dots, f_n)(x) \neq 0$ となる。すなわち, n 階の線形常微分方程式の n 個の解のWronskianはどこかの x で 0 にならなければ, 残りの x でも 0 でなくなる。

3.1.3 行列式の微分

一般に函数成分の行列式の微分は以下のようになる。

函数 $a_{ij}(x)$ を成分とする $n \times n$ 行列を $A(x) = [a_{ij}(x)]$ と書き, その第 j 列を $a_j(x)$ と書き, その (i, j) 余因子を $\Delta_{ij}(x)$ と書き, $\Delta_{ij}(x)$ を (i, j) 成分とする $n \times n$ 行列を $\text{Delta}(x) = [\text{Delta}_{ij}(x)]$ と書く。 $\det A(x) \neq 0$ であると仮定する。このとき, 行列 X の転置を X^T と表すと,

$$\Delta(x)^T = \det(A(x))A(x)^{-1}$$

なので, 以下が成立する:

$$\begin{aligned} \frac{d}{dx} \det A(x) &= \sum_{j=1}^n \det[a_1(x), \dots, a'_j(x), \dots, a_n(x)] \\ &= \sum_{j=1}^n \sum_{i=1}^n \Delta_{ij}(x)a'_{ij}(x) = \text{tr}\left(\Delta(x)^T \frac{dA(x)}{dx}\right) \\ &= \text{tr}\left(A(x)^{-1} \frac{dA(x)}{dx}\right) \det(A(x)). \end{aligned}$$

3.2 Vandermondeの行列式

次の公式が成立している:

$$\begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix} = \prod_{1 \leq i < j \leq n} (x_j - x_i).$$

左辺をVandermonde(ファンデルモンド, ヴァンデルモンド)の行列式と呼び, 右辺を差積と呼ぶのであった.

この公式の証明は n に関する帰納法で以下のように遂行される. $n = 1$ の場合には両辺が 1 になるので成立している. n の場合に成立していると仮定する. このとき,

$$\begin{aligned} & \begin{vmatrix} 1 & 1 & \cdots & 1 & 1 \\ x_1 & x_2 & \cdots & x_n & x_{n+1} \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} & x_{n+1}^{n-1} \\ x_1^n & x_2^n & \cdots & x_n^n & x_{n+1}^n \end{vmatrix} \\ &= \begin{vmatrix} 0 & 0 & \cdots & 0 & 1 \\ x_1 - x_{n+1} & x_2 - x_{n+1} & \cdots & x_n - x_{n+1} & x_{n+1} \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{n-1} - x_{n+1}^{n-1} & x_2^{n-1} - x_{n+1}^{n-1} & \cdots & x_n^{n-1} - x_{n+1}^{n-1} & x_{n+1}^{n-1} \\ x_1^n - x_{n+1}^n & x_2^n - x_{n+1}^n & \cdots & x_n^n - x_{n+1}^n & x_{n+1}^n \end{vmatrix} \\ &= \begin{vmatrix} 0 & 0 & \cdots & 0 & 1 \\ x_1 - x_{n+1} & x_2 - x_{n+1} & \cdots & x_n - x_{n+1} & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{n-1} - x_{n+1}^{n-1} & x_2^{n-1} - x_{n+1}^{n-1} & \cdots & x_n^{n-1} - x_{n+1}^{n-1} & 0 \\ x_1^n - x_{n+1}^n & x_2^n - x_{n+1}^n & \cdots & x_n^n - x_{n+1}^n & 0 \end{vmatrix} \\ &= (x_1 - x_{n+1})(x_2 - x_{n+1}) \cdots (x_n - x_{n+1}) \\ &\quad \times \begin{vmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{n-2} + \cdots + x_{n+1}^{n-2} & x_2^{n-2} + \cdots + x_{n+1}^{n-2} & \cdots & x_2^{n-2} + \cdots + x_{n+1}^{n-2} & 0 \\ x_1^{n-1} + x_1^{n-2}x_{n+1} + \cdots + x_{n+1}^{n-1} & x_2^{n-1} + x_2^{n-2}x_{n+1} + \cdots + x_{n+1}^{n-1} & \cdots & x_2^{n-1} + x_2^{n-2}x_{n+1} + \cdots + x_{n+1}^{n-1} & 0 \end{vmatrix} \\ &= (x_1 - x_{n+1})(x_2 - x_{n+1}) \cdots (x_n - x_{n+1}) \\ &\quad \times \begin{vmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 0 \\ x_1 & x_2 & \cdots & x_n & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} & 0 \end{vmatrix} \\ &= (x_1 - x_{n+1})(x_2 - x_{n+1}) \cdots (x_n - x_{n+1})(-1)^n \begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix} \\ &= (x_{n+1} - x_1)(x_{n+1} - x_2) \cdots (x_{n+1} - x_n) \begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix} \\ &= (x_{n+1} - x_1)(x_{n+1} - x_2) \cdots (x_{n+1} - x_n) \prod_{1 \leq i < j \leq n} (x_j - x_i) \\ &= \prod_{1 \leq i < j \leq n+1} (x_j - x_i). \end{aligned}$$

1つ目の等号では第 $n + 1$ 列を他の列から引き, 2つ目の等号では第1行の x_{n+1}^{i-1} 倍を第 $i = 2, \dots, n + 1$ 行から引いた. 3つ目の等号は第 $j = 1, \dots, n$ 列が $x_j - x_{n+1}$ で割り切ることを使った. 4つ目の等号では, 第 n 行の x_{n+1} 倍を第 $n + 1$ から引き, それと同様の操作を下から順番に行なった. 残りは易しい計算である.

注意: 以上では行列式に関する操作による帰納法でVandermondeの行列式が差積に等しくなることを証明したが, 他にも様々な証明が

ある. 例えば, Vandermondeの行列式が $x_i = x_j$ ($i \neq j$) のとき0になることから, $x_j - x_i$ ($i < j$) で割り切ることを使う証明も有名である. 以下を見よ. \square

3.2.1 Vandermondeの行列式が差積になることの別証明

Vandermondeの行列式を $f(x) = f(x_1, \dots, x_n)$ と表すこととする:

$$f(x) = f(x_1, \dots, x_n) = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{vmatrix}.$$

Vandermondeの行列式 $f(x)$ は x_1, \dots, x_n の多項式になる.

$i \neq j$ のとき, x_j に x_i を代入すると $f(x) = 0$ となりので, 剰余定理によって, $f(x)$ は $x_j - x_i$ で割り切れる. ゆえに $f(x)$ は差積

$$\Delta(x) = \Delta(x_1, \dots, x_n) = \prod_{1 \leq i < j \leq n} (x_j - x_i)$$

で割り切れる. Vandermondeの行列式 $f(x)$ も差積 $\Delta(x)$ も x_1, \dots, x_n の $0 + 1 + \cdots + (n-1)$ 次の多項式になる. そのことより, $f(x)$ は $\Delta(x)$ の定数倍になることがわかる. $f(x)$ における $x_2 x_3^2 \cdots x_n^{n-1}$ (対角成分の積)の係数は 1 であり,

$$\begin{aligned} \Delta(x) &= (x_2 - x_1) \\ &\quad (x_3 - x_1)(x_3 - x_2) \\ &\quad \cdots \cdots \cdots \cdots \cdots \cdots \cdots \\ &\quad (x_n - x_1)(x_n - x_2) \cdots (x_n - x_{n-1}) \end{aligned}$$

における $x_2 x_3^2 \cdots x_n^{n-1}$ の係数も 1 になることがわかる. ゆえに $f(x) = \Delta(x)$ である.

問題: [Cauchyの行列式](https://www.google.com/search?q=Proof-of-Cauchy%27s-determinant) (<https://www.google.com/search?q=Proof-of-Cauchy%27s-determinant>)について調べ, それに関する公式と証明を理解せよ. \square

様々な機会に行列式に関する様々な公式について調べておくと, 後でよいことがあるかもしれない. 行列式に関する各種公式は理論的に重要な役目を果たすことが多く, 数学的教養として行列式に関する各種公式について理解しておくことは大事なことである.

3.3 互いに異なる α_j に対する $e^{\alpha_j x}$ 達の一次独立性

$\alpha_j \in \mathbb{C}$ であるとし, $f_j(x) = e^{\alpha_j x}$ とおく. このとき, f_1, \dots, f_n のWronskianは, Vandermondeの行列式が差積に等しくなることより,

$$\begin{aligned} W(f_1, \dots, f_n) &= \begin{vmatrix} e^{\alpha_1 x} & e^{\alpha_2 x} & \cdots & e^{\alpha_n x} \\ \alpha_1 e^{\alpha_1 x} & \alpha_2 e^{\alpha_2 x} & \cdots & \alpha_n e^{\alpha_n x} \\ \vdots & \vdots & & \vdots \\ \alpha_1^{n-1} e^{\alpha_1 x} & \alpha_2^{n-1} e^{\alpha_2 x} & \cdots & \alpha_n^{n-1} e^{\alpha_n x} \end{vmatrix} \\ &= e^{(\alpha_1 + \cdots + \alpha_n)x} \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i). \end{aligned}$$

となる. ゆえに, $\alpha_1, \dots, \alpha_n$ が互いに異なるならば, このWronskianは決して 0 にならないので, $f_j(x) = e^{\alpha_j x}$ ($j = 1, \dots, n$) は一次独立になる.

3.4 互いに異なる α_j に対する(多項式函数)× $e^{\alpha_j x}$ 達の一次独立性

$f_1(x), \dots, f_r(x)$ は複素係数の多項式函数であるとし, $\alpha_1, \dots, \alpha_r \in \mathbb{C}$ は互いに異なると仮定する. このとき,

$$f_1(x)e^{\alpha_1 x} + \cdots + f_r(x)e^{\alpha_r x} = 0 \quad (x \in \mathbb{R})$$

ならば $f_1 = \cdots = f_r = 0$ となる. このことから, $e^{\alpha_j x}, x e^{\alpha_j x}, x^2 e^{\alpha_j x}, \dots$ 達が一次独立であることがわかる.

証明: 一致の定理より, すべての実数 x について $f_1(x)e^{\alpha_1 x} + \cdots + f_r(x)e^{\alpha_r x} = 0$ が成立することと, すべての複素数 x についてそれが成立することは同値である. ゆえに f_1, \dots, f_r の中に 0 でないものが存在すると仮定して,

$$f_1(z)e^{\alpha_1 z} + \cdots + f_r(z)e^{\alpha_r z} = 0 \quad (z \in \mathbb{C}) \tag{*}$$

とならないことを示せばよい. $f_1, \dots, f_r \neq 0$ かつ $\max\{|\alpha_1|, \dots, |\alpha_r|\} = |\alpha_r|$ と仮定してよい. (0 になる f_j が存在するなら除いて考える.) $\alpha_1, \dots, \alpha_r$ が互いに異なるならば, $\alpha_r = |\alpha_r|e^{i\theta}$, $\theta \in \mathbb{R}$ とおくとき,

$$\operatorname{Re}(\alpha_1 e^{-i\theta}), \dots, \operatorname{Re}(\alpha_{r-1} e^{-i\theta}) < \operatorname{Re}(\alpha_r e^{-i\theta})$$

となる。これでもしも (*) が成立しているならば、絶対値が十分大きなすべての $z \in \mathbb{C}$ について、

$$\frac{f_1(z)}{f_r(z)} e^{(\alpha_1 - \alpha_r)z} + \dots + \frac{f_{r-1}(z)}{f_r(z)} e^{(\alpha_{r-1} - \alpha_r)z} + 1 = 0$$

となる。しかし、 $z = te^{-i\theta}$, $t \in \mathbb{R}$ とおいて、 $t \rightarrow \infty$ とすると、 $j = 1, \dots, r-1$ について、

$$\left| \frac{f_j(z)}{f_r(z)} e^{(\alpha_j - \alpha_r)z} \right| = \left| \frac{f_j(z)}{f_r(z)} \right| e^{(\operatorname{Re}(\alpha_1 e^{-i\theta}) - \operatorname{Re}(\alpha_r e^{-i\theta}))t} \rightarrow 0$$

となるので、 $1 = 0$ となって矛盾する。これで(*)が成立していないことがわかった。□

注意: ツイッターにおける発言

- <https://twitter.com/genkuroki/status/1118680294343659520> (<https://twitter.com/genkuroki/status/1118680294343659520>)

における注意も参考せよ。以下の画像中の行列式の公式(の一般的の場合)を使っても、互いに異なる α_j 達に対する $e^{\alpha_j x}, xe^{\alpha_j x}, x^2 e^{\alpha_j x}, \dots$ 達の一次独立性を(より代数的に)証明することができる。以下の行列式公式が成立している。コンピューターの数式処理によるさらに下の方の計算も参考せよ。□

3.5 Vandermondeの行列式の公式の一般化

$n \times m$ 行列 $P_{n,m}(\alpha)$ を次のように定める:

$$P_{n,m}(\alpha) = \begin{bmatrix} \binom{i-1}{j-1} \alpha^{i-j} \\ 1 & 0 & 0 & \cdots & 0 \\ \alpha & 1 & 0 & \ddots & \vdots \\ \alpha^2 & 2\alpha & 1 & \ddots & 0 \\ \alpha^3 & 3\alpha^2 & 3\alpha & \ddots & 0 \\ \alpha^4 & 4\alpha^3 & 6\alpha^2 & \ddots & 1 \\ \alpha^5 & 5\alpha^4 & 10\alpha^3 & \ddots & \binom{m}{m-1}\alpha \\ \vdots & \vdots & \vdots & & \vdots \\ \alpha^{n-1} & (n-1)\alpha^{n-2} & \binom{n-1}{2}\alpha^{n-3} & \cdots & \binom{n-1}{m-1}\alpha^{n-m} \end{bmatrix}.$$

$P_{n,m}(\alpha)$ の第 j 列は第 1 列を α で $j-1$ 回微分して、 $(j-1)!$ で割ったものに等しい。

さらに、 $m_1 + \dots + m_s = n$ であるとし、 $n \times n$ 行列 P を

$$P = [P_{n,m_1}(\alpha_1), \dots, P_{n,m_s}(\alpha_s)]$$

と定める。このとき、

$$\det P = \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i)^{m_i m_j}$$

が成立する。

この公式の証明は筆者による2004年前期の線形代数の演習問題集に収録されているが、インターネット上では未公開のままである(であったと思う)。証明の方針は簡単で、Vandermondeの行列式が差積に等しいという公式を適切な回数だけ偏微分して、異なる変数を同じにすればよい。

筆者の他に証明をインターネット上で公開している人を検索して探してみたら、同じ方針による証明が次の文献で公開されていることがわかった:

- Garret Sobczyk. Generalized Vandermonde determinants and applications. Aportaciones Matematicas, Serie Comunicaciones 30 (2002) 41--53. ([PDF](http://www.garretstar.com/secciones/publications/docs/generalized_Vandermonde.pdf) (http://www.garretstar.com/secciones/publications/docs/generalized_Vandermonde.pdf))

3.5.1 互いに異なる α_j に対する(多項式函数) $\times e^{\alpha_j x}$ 達の一次独立性の別証明

一般にLeibniz則 (https://en.wikipedia.org/wiki/General_Leibniz_rule) より

$$(fg)^{(i-1)} = \sum_{k=0}^{i-1} \binom{i-1}{k} f^{(k)} g^{(i-1-k)}$$

なので

$$\left(\frac{x^{j-1}}{(j-1)!} e^{\alpha x} \right)^{(i-1)} = \sum_{k=0}^{i-1} \binom{i-1}{k} \frac{x^{j-1-k}}{(j-1-k)!} \alpha^{i-1-k} e^{\alpha x}.$$

この $x = 0$ での値は $\binom{i-1}{j-1} \alpha^{i-j}$ になり、 $P_{n,m}(\alpha)$ の (i, j) 成分に等しくなる。ゆえに Wronskian の $x = 0$ での値を Vandermonde の行列式の公式の一般化を用いて評価することによって、互いに異なる α_j 達に対する $e^{\alpha_j x}, xe^{\alpha_j x}, x^2 e^{\alpha_j x}, \dots$ 達の一次独立性の証明が得られる。

3.5.2 WolframAlphaによる一般化されたVandermondeの行列式の計算例

- [WolframAlpha \(https://www.wolframalpha.com/\)](https://www.wolframalpha.com/)

In [3]: 1 showing("image/jpeg", "images/generalized-Vandermonde-1.jpg", scale="70%")

The screenshot shows the WolframAlpha interface with the input query: `factor det{{1,0,1,1},{a,1,c,d,f},{a^2,2a,c^2,d^2,f^2},{a^3,3a^2,c^3,d^3,f^3},{a^4,4a^3,c^4,d^4,f^4}}`. The result section displays the determinant as a factored polynomial: $-(a - c)^2 (a - d)^2 (a - f)^2 (c - d) (c - f) (d - f)$.

In [4]: 1 showing("image/jpeg", "images/generalized-Vandermonde-2.jpg", scale="70%")

The screenshot shows the WolframAlpha interface with the input query: `factor det{{1,0,0,1,1},{a,1,0,d,f},{a^2,2a,1,d^2,f^2},{a^3,3a^2,3a,d^3,f^3},{a^4,4a^3,6a^2,d^4,f^4}}`. The result section displays the determinant as a factored polynomial: $-(a - d)^3 (a - f)^3 (d - f)$.

In [5]: 1 showing("image/jpeg", "images/generalized-Vandermonde-3.jpg", scale="70%")

The screenshot shows the WolframAlpha interface with the input query: `factor det{{1,0,0,1,0},{a,1,0,d,1},{a^2,2a,1,d^2,2d},{a^3,3a^2,3a,d^3,3d^2},{a^4,4a^3,6a^2,d^4,4d^3}}`. The result section displays the determinant as a single term: $(a - d)^6$.

In [6]:

```
1 showing("image/jpeg", "images/generalized-Vandermonde-4.jpg", scale="70%")
```

factor $\det\{1, 0, 1, 0, 1\}, \{a, 1, c, 1, f\}, \{a^2, 2a, c^2, 2c, f^2\}, \{a^3, 3a^2, c^3, 3c^2, f^3\}, \{a^4, 4a^3, c^4, 4c^3, f^4\}$

Input interpretation

$$\text{factor} \begin{vmatrix} 1 & 0 & 1 & 0 & 1 \\ a & 1 & c & 1 & f \\ a^2 & 2a & c^2 & 2c & f^2 \\ a^3 & 3a^2 & c^3 & 3c^2 & f^3 \\ a^4 & 4a^3 & c^4 & 4c^3 & f^4 \end{vmatrix}$$

Result

$$(a - c)^4 (a - f)^2 (c - f)^2$$

3.5.3 SymPyによる一般化されたVandermondeの行列式の計算例

In [7]:

```
1 function Pblock(a, n, m)
2     P = Array{Sym, 2}(undef, n, m)
3     for j in 1:m
4         for i in 1:n
5             if i < j
6                 P[i,j] = 0
7             elseif i == j
8                 P[i,j] = 1
9             else
10                P[i,j] = binomial(i-1, j-1)*a^(i-j)
11            end
12        end
13    end
14    P
15 end
16
17 function Pmatrix(as, ms)
18     n = sum(ms)
19     hcat(Pblock(as[i], n, ms[i]) for i in 1:length(as))...
20 end
```

Out[7]: Pmatrix (generic function with 1 method)

In [8]:

```
1 @vars a b c d
2 V = Pmatrix([a, b, c, d], [1, 1, 1, 1])
```

Out[8]:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ a & b & c & d \\ a^2 & b^2 & c^2 & d^2 \\ a^3 & b^3 & c^3 & d^3 \end{bmatrix}$$

In [9]:

```
1 @time V.det().factor()
```

1.611147 seconds (2.25 M allocations: 111.579 MiB, 3.20% gc time)

Out[9]: $(a - b)(a - c)(a - d)(b - c)(b - d)(c - d)$

In [10]:

```
1 @vars a b c d
2 P = Pmatrix([a, b, c, d], [3, 2, 1, 1])
```

Out[10]:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ a & 1 & 0 & b & 1 & c & d \\ a^2 & 2a & 1 & b^2 & 2b & c^2 & d^2 \\ a^3 & 3a^2 & 3a & b^3 & 3b^2 & c^3 & d^3 \\ a^4 & 4a^3 & 6a^2 & b^4 & 4b^3 & c^4 & d^4 \\ a^5 & 5a^4 & 10a^3 & b^5 & 5b^4 & c^5 & d^5 \\ a^6 & 6a^5 & 15a^4 & b^6 & 6b^5 & c^6 & d^6 \end{bmatrix}$$

In [11]: 1 @time P.det().factor()

5.846432 seconds (161 allocations: 6.672 KiB)

Out[11]: $-(a - b)^6(a - c)^3(a - d)^3(b - c)^2(b - d)^2(c - d)$

In [12]: 1 @vars a b c
2 Q = Pmatrix([a, b, c], [4, 2, 1])

Out[12]:
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ a & 1 & 0 & 0 & b & 1 & c \\ a^2 & 2a & 1 & 0 & b^2 & 2b & c^2 \\ a^3 & 3a^2 & 3a & 1 & b^3 & 3b^2 & c^3 \\ a^4 & 4a^3 & 6a^2 & 4a & b^4 & 4b^3 & c^4 \\ a^5 & 5a^4 & 10a^3 & 10a^2 & b^5 & 5b^4 & c^5 \\ a^6 & 6a^5 & 15a^4 & 20a^3 & b^6 & 6b^5 & c^6 \end{bmatrix}$$

In [13]: 1 @time Q.det().factor()

0.960101 seconds (161 allocations: 6.672 KiB)

Out[13]: $(a - b)^8(a - c)^4(b - c)^2$

In []:

1

FFTを用いた偏微分方程式の数値解法(in-place版)

黒木玄

2018-01-23～2019-04-16

- Copyright 2018, 2019 Gen Kuroki
- License: MIT <https://opensource.org/licenses/MIT> (<https://opensource.org/licenses/MIT>)

このノートブックは最初以下の2つのノートブックの続きとして作成された。

- [FFTを用いた熱方程式やKdV方程式などを数値解法](http://nbviewer.jupyter.org/gist/genkuroki/14b05a2cfa172fc5ea351641d4bdda11)
(<http://nbviewer.jupyter.org/gist/genkuroki/14b05a2cfa172fc5ea351641d4bdda11>)
- [2次元配列でのFFTの使い方](http://nbviewer.jupyter.org/gist/genkuroki/26928d4a1ae2e912a3850ed1b31e2941) (<http://nbviewer.jupyter.org/gist/genkuroki/26928d4a1ae2e912a3850ed1b31e2941>)

このノートブックでは in-place でFFTを使うことによってメモリ効率を最適化することを行いたい。

例として空間が1次元の熱方程式, KdV方程式, Schrödinger方程式, Smith方程式を扱う。

目次

1 諸定義

1.1 パッケージの読み込み

1.2 数式を使った解説

1.3 FFT_Data タイプの定義

1.4 プロット用の函数

2 热方程式で以上の定義のテスト

3 KdV方程式の数値解法の最適化

3.1 プロット用函数

3.2 FFTを `FFT * v`, `FFT \ v` の形式で利用した場合

3.3 FFTを in-place で利用した場合

3.4 KdV方程式: 初期条件 sin

3.5 ベンチマークテスト

3.6 KdV方程式: 2-soliton解

4 空間1次元の時間依存のSchrödinger方程式

4.1 Gaussian_packet

4.2 自由粒子

4.3 低い壁

4.4 高い壁

4.5 調和振動子

4.6 $V(x) = -100 \exp(-x^2/100)$

5 Smith方程式

5.1 Smith方程式: 初期条件 sin

5.2 Smith方程式: 初期条件 KdV 2-soliton

1 諸定義

1.1 パッケージの読み込み

```
In [1]: using Base64
using FFTW
using LinearAlgebra
const e = ℯ

using PyPlot

default_figsize = (6.4, 4.8) # このデフォルトサイズは大き過ぎるので
rc("figure", figsize=(3,2.4)) # このように小さくしておく.

using PyCall
const animation = pyimport("matplotlib.animation")

function displayfile(mimetype, file)
    open(file) do f
        base64text = base64encode(f)
        display("text/html", """""")
    end
end
end
```

Out[1]: displayfile (generic function with 1 method)

```
In [2]: 1 @show Sys.CPU_THREADS
2 FFTW.set_num_threads(Sys.CPU_THREADS)
```

Sys.CPU_THREADS = 8

1.2 数式を使った解説

周期 $L = 2\pi K$ を持つ函数を

$$f(x) = \sum_{k=-N/2+1}^{N/2} (-1)^{k-1} a_k e^{2\pi i(k-1)x/L} = \sum_{k=-N/2+1}^{N/2} (-1)^{k-1} a_k e^{i(k-1)x/K}$$

の形の有限Fourier級数でよく近似できていると仮定する。この導函数は

$$f'(x) = \sum_{k=-N/2+1}^{N/2} (-1)^{k-1} \frac{i(k-1)}{K} a_k e^{i(k-1)x/K}$$

になる。すなわち, $f(x)$ を微分する操作は, $-N/2 < k \leq N/2$ のとき a_k に $i(k-1)/K$ をかける操作に対応している。
 $-N/2 < k \leq N/2$ のとき a_k に $i(k-1)/K$ をかける操作は(丸め誤差)を除いて、有限Fourier級数の正確な微分を与えることに注意せよ。この計算方法は小さな h について $(f(x + h/2) - f(x - h/2))/h$ のような方法で微分を求める方法よりも誤差が小さい。

さらに x を

$$\Delta x = \frac{L}{N} = \frac{2\pi K}{N}, \quad x_j = (j-1)\Delta x - \frac{L}{2} = L \left(\frac{j-1}{N} - \frac{1}{2} \right)$$

で離散化したとする。このとき,

$$f(x_j) = \sum_{k=-N/2+1}^{N/2} a_k e^{i(k-1)(j-1)/N}.$$

これは, $a_{k+N} = a_k$ という仮定のもとで

$$a_{k+N} e^{i((k+N)-1)(j-1)/N} = a_k e^{i(k-1)(j-1)/N}$$

であり, $k = -N/2 + 1, -N/2 + 2, \dots, 0$ が $k + N = N/2 + 1, N/2 + 2, \dots, N$ に対応するので,

$$f(x_j) = \sum_{k=1}^N a_k e^{i(k-1)(j-1)/N}$$

と書き直される。これはJulia言語の離散Fourier変換のスタイルに一致している。

この表示のもとで, $f(x)$ を微分する操作は, $1 \leq k \leq N/2$ のとき $i(k-1)/N$ を a_k にかけ, $N/2 + 1 \leq k \leq N$ のとき $i(k-1-N)/N$ を a_k にかける操作に対応している。下の方で定義される `FFT_Data` における `D` を成分ごとにかける操作はまさにこの操作になっている。

In [3]: 1 ?ifft

search: ifft ifft! ifftshift irfft plan_ifft plan_irfft! Cptrdiff_t plan_irfft

Out[3]: ifft(A [, dims])

Multidimensional inverse FFT.

A one-dimensional inverse FFT computes

$$\text{IDFT}(A)[k] = \frac{1}{\text{length}(A)} \sum_{n=1}^{\text{length}(A)} \exp\left(+i \frac{2\pi(n-1)(k-1)}{\text{length}(A)}\right) A[n].$$

A multidimensional inverse FFT simply performs this operation along each transformed dimension of A .

1.3 FFT_Data タイプの定義

`roi(a, b, N)` (right-open interval $[a, b)$) は閉区間 $[a, b]$ を N 等分して両端の点も合わせて $N + 1$ 個の点を得た後に右端の点を除いたものになる。右半開区間 $[a, b)$ の N 等分だと思ってよい。

`x_axis(L, N)` は $[-L/2, L/2)$ の N 等分になり, x 軸の離散化とみなされる。

`k_axis(N)` は波数空間(波数軸)の離散化を意味する。上の数式を使った解説より, `k_axis(N)` は右半開区間 $[0, N/2)$ と $[-N/2, 0)$ の離散化をその順序で連結したものにしなければいけない。

`o = FFT_Data(K, N)` 型のとき, `o` は $K = 2\pi K$ のときの周期境界条件が課された $[-L/2, L/2)$ の N 等分を x 軸の離散化とみなしたときの, 高速Fourier変換関係のデータになる。

- `o.K` は K になる。
- `o.L` は x 軸方向の周期の長さ $2\pi K$ になる。
- `o.N` は N になる。 $[-L/2, L/2)$ が N 等分される。
- `o.x` は `x_axis(L, N)` になる。
- `o.k` は `k_axis(N)` になる。
- `o.D` は `im .* k ./ K` すなわちそれを成分ごとにかける操作は波数空間で実現された $\partial/\partial x$ になる。
- `o.D2` を成分ごとにかける操作は波数空間で実現された $(\partial/\partial x)^2$ になる。
- `o.D3` を成分ごとにかける操作は波数空間で実現された $(\partial/\partial x)^3$ になる。
- `o.D4` を成分ごとにかける操作は波数空間で実現された $(\partial/\partial x)^4$ になる。
- `o.FFT` は以上の設定における高速Fourier変換の「プラン」になる。

In [4]:

```

1 # right-open interval [a,b)
2 #
3 roi(a, b, N) = collect(range(a, b-(b-a)/N, length=N))
4
5 # x axis (Assume the periodic boundary condition with period L)
6 #
7 x_axis(L, N) = roi(-L/2, L/2, N)
8
9 # k axis (k = the wave number)
10 #
11 function k_axis(N)
12     @assert iseven(N)
13     Ndiv2 = div(N,2)
14     vcat(roi(0,Ndiv2,Ndiv2), roi(-Ndiv2,0,Ndiv2))
15 end
16
17 # FFT Data
18 #
19 mutable struct FFT_Data{T<:Real, S}
20     K::T
21     L::T
22     N::Int64
23     x::Array{T,1}
24     k::Array{T,1}
25     D::Array{Complex{T},1}
26     D2::Array{Complex{T},1}
27     D3::Array{Complex{T},1}
28     D4::Array{Complex{T},1}
29     FFT::S
30 end
31
32 # FFT | (D.^m .* (FFT * u)) = (d/dx)^m u
33 #
34 function FFT_Data(K, N)
35     L = 2π*K
36     x = x_axis(L, N)
37     k = k_axis(N)
38     D = im .* k ./ K
39     FFT = plan_fft(Array{Complex{Float64},1}(undef, N))
40     FFT_Data(Float64(K), L, N, x, k, D, D.^2, D.^3, D.^4, FFT)
41 end

```

Out[4]: FFT_Data

Julia言語におけるFFTの使い方については以下を参照せよ.

In [5]: 1 ?plan_fft

search: plan_fft plan_fft! plan_rfft plan_ifft plan_bfft plan_ifft! plan_bfft!

Out[5]: plan_fft(A [, dims]; flags=FFTW.ESTIMATE, timelimit=Inf)

Pre-plan an optimized FFT along given dimensions (`dims`) of arrays matching the shape and type of `A` . (The first two arguments have the same meaning as for [fft_\(@ref\)](#).) Returns an object `P` which represents the linear operator computed by the FFT, and which contains all of the information needed to compute `fft(A, dims)` quickly.

To apply `P` to an array `A` , use `P * A` ; in general, the syntax for applying plans is much like that of matrices. (A plan can only be applied to arrays of the same size as the `A` for which the plan was created.) You can also apply a plan with a preallocated output array `Â` by calling `mul!(Â, plan, A)` . (For `mul!` , however, the input array `A` must be a complex floating-point array like the output `Â` .) You can compute the inverse-transform plan by `inv(P)` and apply the inverse plan with `P \ Â` (the inverse plan is cached and reused for subsequent calls to `inv` or `\`), and apply the inverse plan to a pre-allocated output array `A` with `ldiv!(A, P, Â)` .

The `flags` argument is a bitwise-or of FFTW planner flags, defaulting to `FFTW.ESTIMATE` . e.g. passing `FFTW.MEASURE` or `FFTW.PATIENT` will instead spend several seconds (or more) benchmarking different possible FFT algorithms and picking the fastest one; see the FFTW manual for more information on planner flags. The optional `timelimit` argument specifies a rough upper bound on the allowed planning time, in seconds. Passing `FFTW.MEASURE` or `FFTW.PATIENT` may cause the input array `A` to be overwritten with zeros during plan creation.

`plan_fft!_(@ref)` is the same as `plan_fft_(@ref)` but creates a plan that operates in-place on its argument (which must be an array of complex floating-point numbers). `plan_ifft_(@ref)` and so on are similar but produce plans that perform the equivalent of the inverse transforms `ifft_(@ref)` and so on.

1.4 プロット用の函数

虚部が至る所ほぼ 0 なら虚部をプロットしない。

```
1 # nearly zero
2 #
3 function imagisnealyzero(z)
4     maximum(abs, imag(z))/(maximum(abs, real(z)) + 1e-15) < 1e-3
5 end
6
7 # plot complex number array u on x
8 #
9 function plot_u(x, u)
10    plot(x, real(u), label="Re", lw=0.8)
11    imagisnealyzero(u) || plot(x, imag(u), label="Im", lw=0.8, ls="--")
12    grid(ls=":")
13    xticks(fontsize=8)
14    yticks(fontsize=8)
15 end
```

Out[6]: plot_u (generic function with 1 method)

2 热方程式で以上の定義のテスト

この節では次の热方程式を x 方向に関する周期境界条件のもとで数値的に解こう:

$$u_t = u_{xx}.$$

この方程式は形式的には次のように解ける: $u(t) : x \mapsto u(t, x)$ について,

$$u(t) = \exp\left(t\left(\frac{\partial}{\partial x}\right)^2\right)u(0).$$

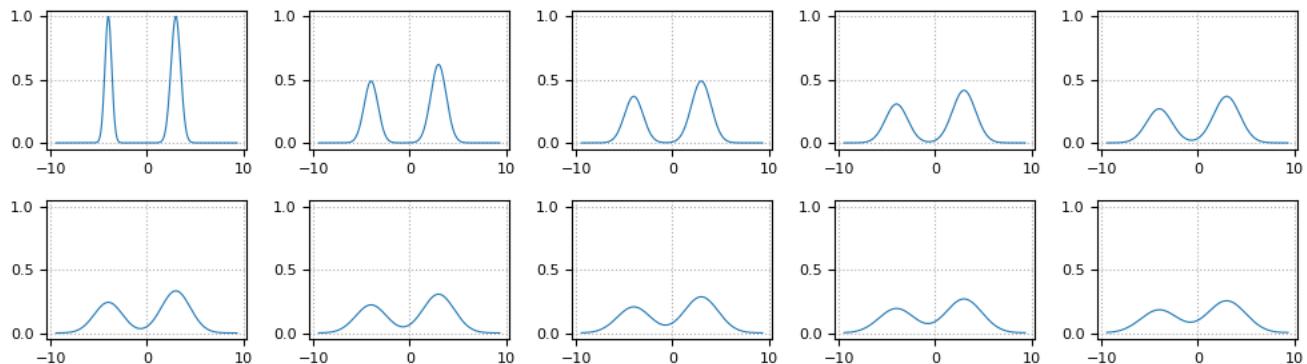
以下では x 方向について離散化し, $(\partial/\partial x)^2$ を高速Fourier変換(FFT)を使って実現する.

In [7]:

```

1 # Application - the solution of the heat equation  $u_t = u_{xx}$ 
2 #
3 #      $u(t) = \exp(t (\partial/\partial x)^2) u(0)$ 
4 #
5 sol_of_heat_eq(o::FFT_Data, u0, t) = o.FFT \ (exp.(t .* o.D2) .* (o.FFT * u0))
6
7 N = 2^8
8 K = 3
9 o = FFT_Data(K, N);
10
11 f0(x) = exp(-4(x+4)^2) + exp(-2(x-3)^2)
12 u0 = f0.(o.x)
13 t = roi(0,2,10)
14 @time u = [sol_of_heat_eq(o, u0, t) for t in t]
15
16 figure(figsize=(10, 3))
17 for j in 1:10
18     subplot(2,5,j)
19     plot_u(o.x, u[j])
20     ylim(-0.05,1.05)
21 end
22 tight_layout()

```



0.632198 seconds (1.92 M allocations: 98.374 MiB, 5.18% gc time)

3 KdV方程式の数値解法の最適化

この節では次の形でのKdV方程式を x 方向に関する周期境界条件のもとで数値的に解こう:

$$u_t = -u_{xxx} - 3(u^2)_x = -u_{xxx} - 6uu_x.$$

この方程式に従う時間発展は Δt が微小なとき次のように近似的に書き直される:

$$\begin{aligned} u(t, x) &\mapsto \tilde{u}(t, x) = \exp\left(-\Delta t \left(\frac{\partial}{\partial x}\right)^3\right) u(t) \\ &\mapsto u(t + \Delta t, x) = \tilde{u}(t, x) - 3\Delta t \frac{\partial}{\partial x}(\tilde{u}(t, x)^2). \end{aligned}$$

以下では x 方向について離散化し, $(\partial/\partial x)^3$ や $\partial/\partial x$ を高速Fourier変換(FFT)を使って実現する.

3.1 プロット用函数

Out[8]: anim KdV (generic function with 1 method)

3.2 FFTを $\text{FFT} * v$, $\text{FFT} \setminus v$ の形式で利用した場合

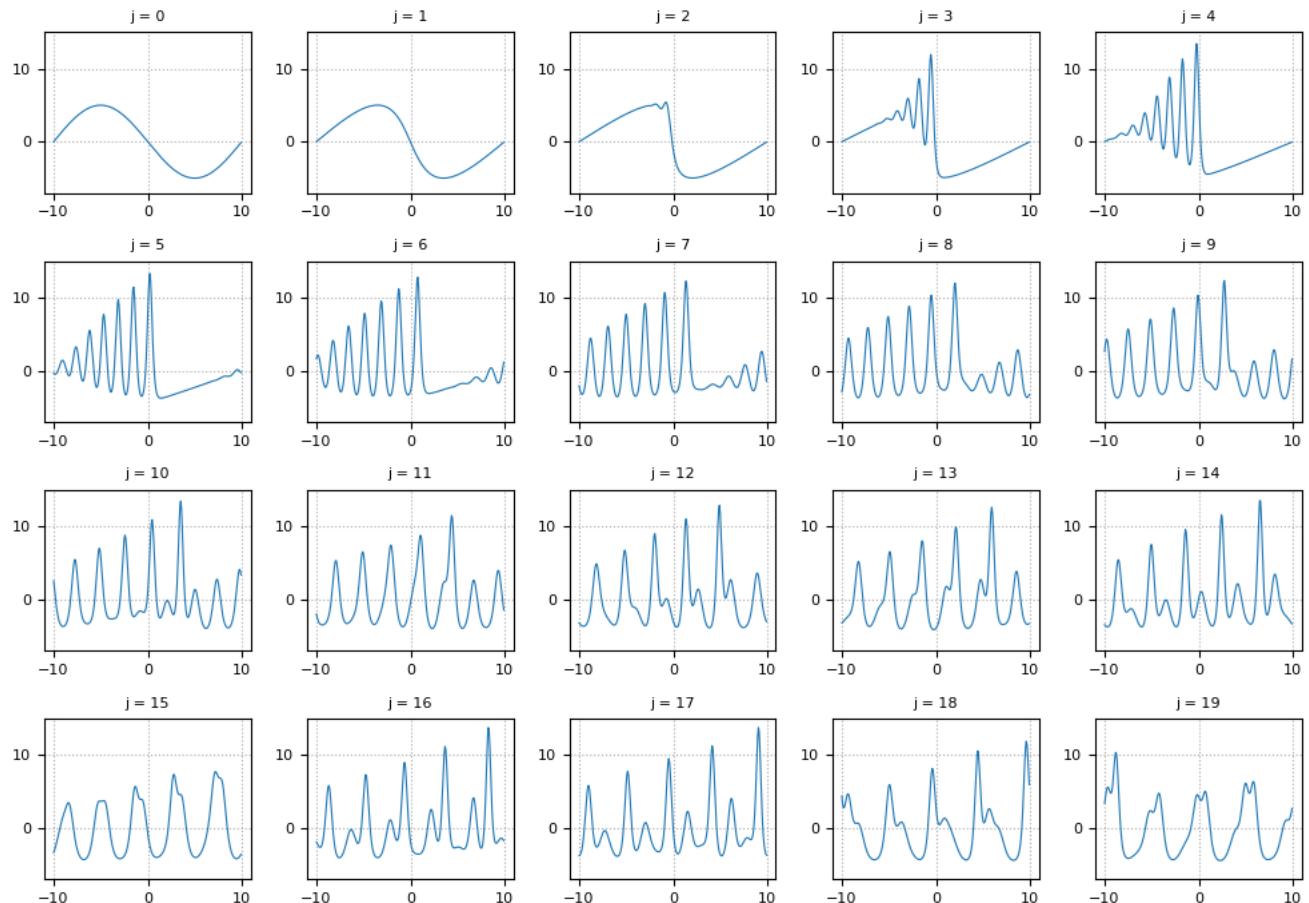
In [9]:

```

1 # FFT*v and FFT|v version
2
3 function update_KdV(o::FFT_Data, u, Δt, niters)
4     v = o.FFT * u
5     for iter in 1:niters
6         v .= exp.(-Δt .* o.D3) .* v
7         v .-= 3.0 .* Δt .* o.D .* (o.FFT * (o.FFT \ v).^2)
8     end
9     o.FFT \ v
10 end
11
12 ▼ function solve_KdV(o::FFT_Data{T}, f0, tmax) where T
13     Δt = 1/o.N^2
14     skip = floor(Int, 0.005/Δt)
15     t = 0:skip*Δt:tmax
16     M = length(t)
17     u = Array{Complex{T},2}(undef, o.N, M)
18     u[:,1] = Complex.(f0.(o.x))
19     for j in 2:M
20         u[:,j] = update_KdV(o, @view(u[:,j-1]), Δt, skip)
21     end
22     return real(u), t
23 end
24
25 N = 2^8
26 K = 20/(2π)
27 o = FFT_Data(K, N);
28
29 f0(x) = -5*sin(π*x/10)
30 Δt = 1/N^2
31 skip = 10*floor(Int, 0.005/Δt)
32
33 tmax = 1.0
34 @time u, t = solve_KdV(o, f0, tmax)
35
36 sleep(0.1)
37 plot_KdV(o, u, t)

```

8.354966 seconds (2.12 M allocations: 888.322 MiB, 2.08% gc time)



3.3 FFTを in-place で利用した場合

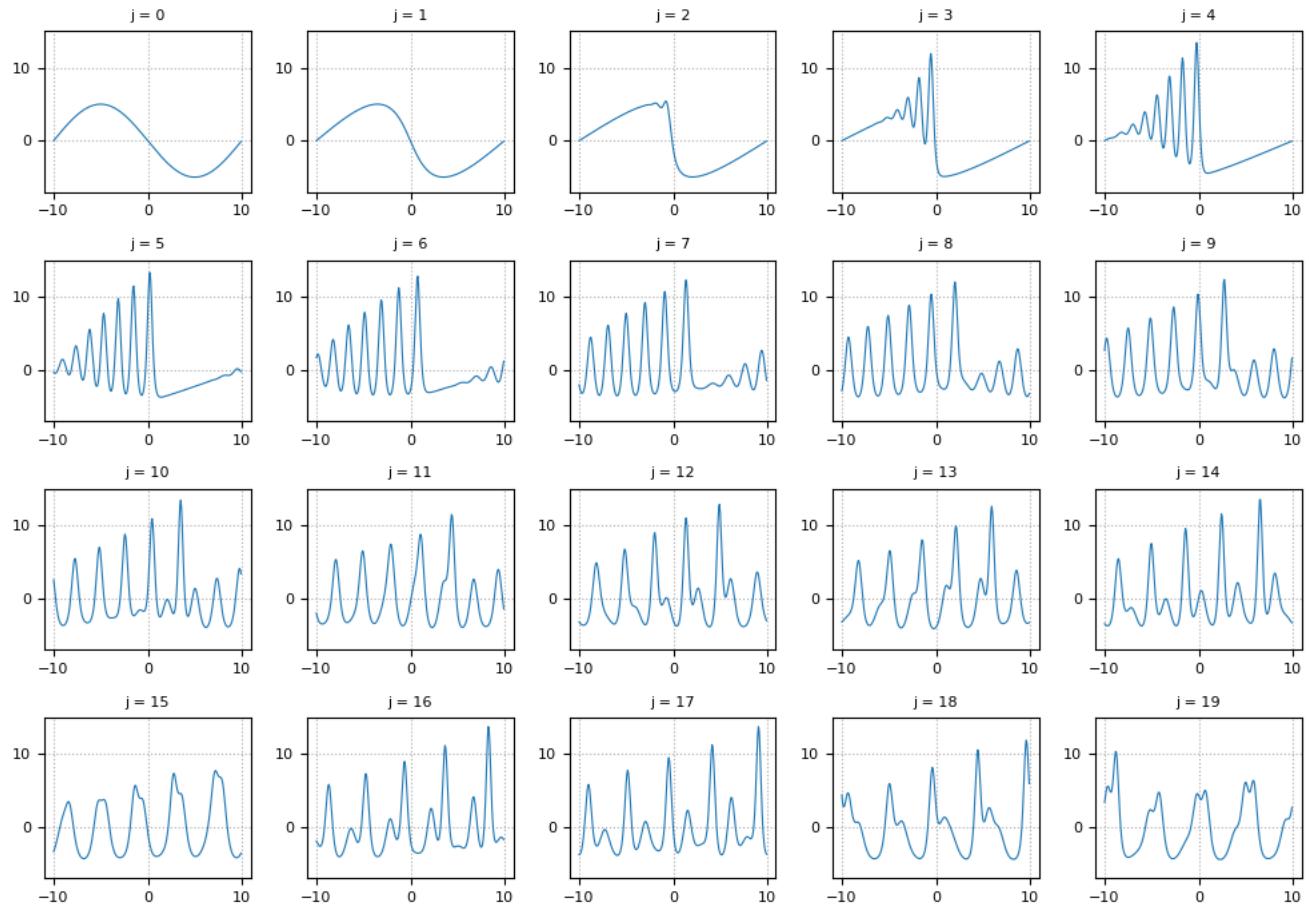
In [10]:

```

1  function update_KdV!(o::FFT_Data, u1, u0, Δt, nitors)
2      v = similar(u0)
3      v_tmp = similar(u0)
4      u_tmp = similar(u0)
5
6      mul!(v, o.FFT, u0)
7      for iter in 1:nitors
8          # v .= exp.(-Δt .* o.D3) .* v
9          v .= exp.(-Δt .* o.D3) .* v
10         # v .-= 3.0 .* Δt .* o.D .* (o.FFT * (o.FFT | v).^2)
11         ldiv!(u_tmp, o.FFT, v)
12         u_tmp .= u_tmp
13         mul!(v_tmp, o.FFT, u_tmp)
14         v .-= 3.0 .* Δt .* o.D .* v_tmp
15     end
16     ldiv!(u1, o.FFT, v)
17 end
18
19
20 function solve_KdV_inplace(o::FFT_Data{T}, f0, tmax) where T
21     Δt = 1/o.N^2
22     skip = floor(Int, 0.005/Δt)
23     t = 0:skip*Δt:tmax
24     M = length(t)
25     u = Array{Complex{T},2}(undef, o.N, M)
26     u[:,1] = Complex.(f0.(o.x))
27     for j in 2:M
28         update_KdV!(o, @view(u[:,j]), @view(u[:,j-1]), Δt, skip)
29     end
30     return real(u), t
31 end
32
33 N = 2^8
34 K = 20/(2π)
35 o = FFT_Data(K, N);
36
37 f0(x) = -5*sin(π*x/10)
38 Δt = 1/N^2
39 skip = 10*floor(Int, 0.005/Δt)
40
41 tmax = 1.0
42 @time u, t = solve_KdV_inplace(o, f0, tmax)
43
44 sleep(0.1)
45 plot_KdV(o, u, t)

```

7.513852 seconds (855.68 k allocations: 46.708 MiB, 0.22% gc time)



3.4 KdV方程式: 初期条件 sin

以下のアニメーションはKdV方程式

$$u_t = -u_{xxx} - 3(u^2)_x = -u_{xxx} - 6uu_x$$

を初期条件

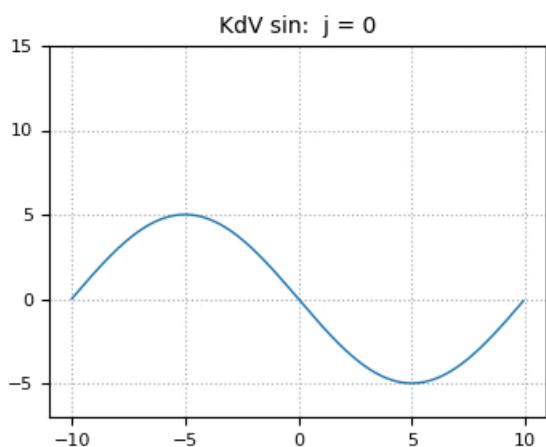
$$u(0, x) = -5 \sin(\pi x/10)$$

のもとで数値的に解いた結果である. sin の初期条件からたくさんのソリトンが放出されている. この結果は

- N.J. Zabusky and M. D. Kruskal, Phy. Rev. Lett., 15, 240 (1965)

による有名な仕事の再現になっている.

```
In [11]: 1 gifname = "KdV_sin"
2 @time anim_KdV(gifname, o, u, t)
```



24.810374 seconds (3.42 M allocations: 179.103 MiB, 0.28% gc time)

3.5 ベンチマークテスト

```
In [12]: 1 using BenchmarkTools
2
3 N = 2^8
4 K = 20/(2π)
5 o = FFT_Data(K, N)
6
7 f0(x) = -5*sin(π*x/10)
8 Δt = 1/N^2
9 skip = 10*floor(Int, 0.005/Δt)
10
11 tmax = 1.4
```

Out[12]: 1.4

```
In [13]: 1 @benchmark u, t = solve_KdV(o, f0, tmax)
```

```
Out[13]: BenchmarkTools.Trial:
memory estimate: 1.10 GiB
allocs estimate: 642611
-----
minimum time: 11.159 s (2.23% GC)
median time: 11.159 s (2.23% GC)
mean time: 11.159 s (2.23% GC)
maximum time: 11.159 s (2.23% GC)
-----
samples: 1
evals/sample: 1
```

```
In [14]: 1 @benchmark u, t = solve_KdV_inplace(o, f0, tmax)
```

```
Out[14]: BenchmarkTools.Trial:
memory estimate: 20.48 MiB
allocs estimate: 276931
-----
minimum time: 10.035 s (0.00% GC)
median time: 10.035 s (0.00% GC)
mean time: 10.035 s (0.00% GC)
maximum time: 10.035 s (0.00% GC)
-----
samples: 1
evals/sample: 1
```

以上のように in-place 版は非 in-place 版よりも計算時間が1割程度短く、in-place版の消費メモリは非in-place版の50分の1である。

3.6 KdV方程式: 2-soliton解

KdV方程式

$$u_t = -u_{xxx} - 3(u^2)_x = -u_{xxx} - 6uu_x$$

の1ソリトン解は $c > 0$ に対する

$$u(t, x) = \frac{c}{2} \operatorname{sech}^2 \frac{\sqrt{c}}{2}(x - ct - a)$$

の形をしている。ここで $\operatorname{sech} x = 1/\cosh x$, $\cosh x = (e^x - e^{-x})/2$ である。 $c > 0$ が大きければ大きいほど、ソリトンは細くて高い山になり、速く進む。

以下はKdV方程式を初期条件

$$u(0, x) = f(16, -8, x) + f(4, -2, x)$$

のもとで数値的に解いた結果である。ここで

$$f(c, a, x) = \frac{c}{2} \operatorname{sech}^2 \frac{\sqrt{c}}{2}(x - a).$$

そのようにして得られたKdV方程式の数値解はKdV方程式の2ソリトン解とはぴったり一致しないが、良い近似にはなっている。KdV方程式は非線形偏微分方程式なので、解は初期条件に線形に依存しない。

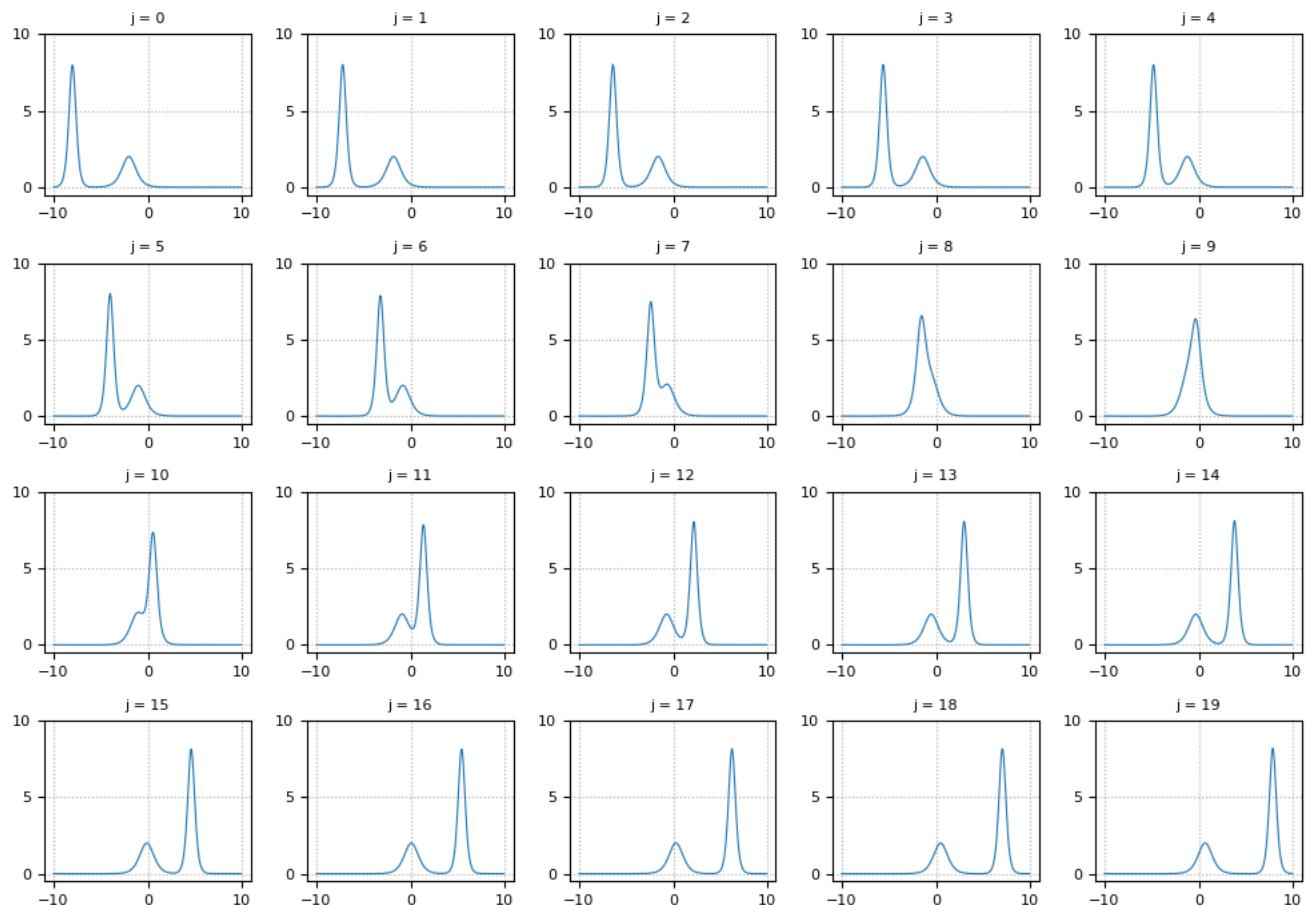
In [15]:

```

1 N = 2^8
2 K = 20/(2π)
3 o = FFT_Data(K, N);
4
5 KdVsoliton(c, a, x) = c/2*(sech(sqrt(c)/2*(x-a)))^2
6 f0(x) = KdVsoliton(16.0, -8.0, x) + KdVsoliton(4.0, -2.0, x)
7 Δt = 1/N^2
8 skip = 10*floor(Int, 0.005/Δt)
9
10 tmax = 1.0
11 @time u, t = solve_KdV_inplace(o, f0, tmax)
12
13 sleep(0.1)
14 plot_KdV(o, u, t, ymin=-0.5, ymax=10.0)

```

7.170397 seconds (386.81 k allocations: 24.449 MiB, 0.88% gc time)

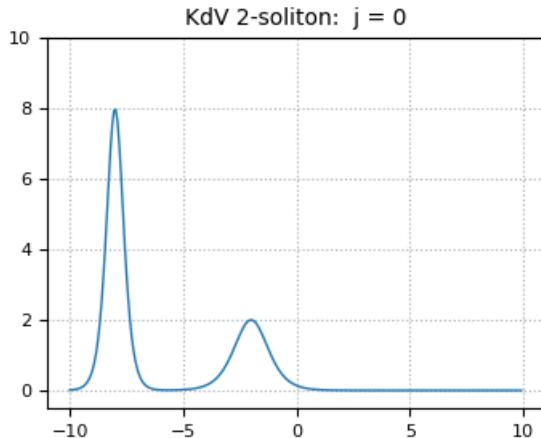


In [16]:

```

1 gifname = "KdV_2-soliton"
2 @time anim_KdV(gifname, o, u, t, ymin=-0.5, ymax=10.0)

```



22.678053 seconds (73.57 k allocations: 9.681 MiB, 0.03% gc time)

速く進む高い山のソリトンと遅く進む低い山のソリトンがぶつかっても壊れずに、2ソリトン状態が維持される。各々のソリトンはまるで粒子のごとく壊れない。

4 空間1次元の時間依存のSchrödinger方程式

以下では次の形のSchrödinger方程式を扱う:

$$i \frac{\partial}{\partial t} \psi = \left[-\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 + V(x) \right] \psi.$$

右辺の [] の内側のoperatorをHamiltonianと呼ぶ。

この微分方程式に従う時間発展は Δt が微小なとき以下で近似される:

$$\psi(t, x) \mapsto \psi(t + \Delta t, x) = \exp(-i \Delta t V(x)) \exp\left(\frac{i}{2} \Delta t \left(\frac{\partial}{\partial x}\right)^2\right) \psi(t, x).$$

$\text{o} = \text{FFT_Data}(K, N)$ のとき, $\text{s} = \text{Schroedinger_Data}(\text{o}, V)$ はFFTに関するデータ o にポテンシャル函数 $V(x)$ のデータを合わせたものになる。 s.v は離散化された x 軸上での $V(x)$ の値になる。

```

In [17]: 1 ▼ # V はポテンシャル函数
2 ▼ #
3 ▼ mutable struct Schroedinger_Data{T, S, R}
4 ▼   o::FFT_Data{T,S}
5   V::R
6 ▼   v::Array{T,1}
7 end
8
9 # FFT_Data o とポテンシャル函数 V から Schroedinger_Data を作成する函数
10 #
11 Schroedinger_Data(o, V) = Schroedinger_Data(o, V, V.(o.x))
12
13 function update_Schroedinger!(s::Schroedinger_Data, u1, u0, Δt, skip)
14     o = s.o
15     U_tmp = similar(u0)
16     u1 .= u0
17
18     for iter in 1:skip
19         # exp.(-im .* Δt .* s.v) .* (o.FFT|(exp.(0.5im .* Δt .* o.D2).* (o.FFT * u)))
20         mul!(U_tmp, o.FFT, u1)
21         U_tmp .= exp.(0.5im .* Δt .* o.D2)
22         ldiv!(u1, o.FFT, U_tmp)
23         u1 .= exp.(-im .* Δt .* s.v)
24     end
25 end
26
27 ▼ function solve_Schroedinger(s::Schroedinger_Data{T,S,R}, u0, tmax; Δt=2π/1000, skip=10) where {T,S,R}
28     o = s.o
29     t = 0:skip*Δt:tmax+10eps()
30     M = length(t)
31 ▼     u = Array{Complex{T},2}(undef, o.N, M)
32 ▼     u[:,1] = u0
33     for j in 2:M
34         update_Schroedinger!(s, @view(u[:,j]), @view(u[:,j-1]), Δt, skip)
35     end
36     return u, t
37 end
38
39 function plot_complexvalued(x, u, ymin, ymax)
40     plot(o.x, real(u), lw=0.8)
41     plot(o.x, imag(u), lw=0.8, ls="--")
42     ylim(ymin, ymax)
43     grid(ls=":")
44     xticks(fontsize=8)
45     yticks(fontsize=8)
46 end
47
48 function plot_Schroedinger(s::Schroedinger_Data, u, t; thin=10, ymin=-1.1, ymax=1.1)
49     nplots = (length(t)-1)÷thin + 1
50     o = s.o
51     nrows = div(nplots+3, 5)
52
53     figure(figsize=(4, 1.75))
54     subplot(121)
55     plot_complexvalued(o.x, u[:,1], ymin, ymax)
56     title("initial state", fontsize=9)
57     subplot(122)
58     plot_u(o.x, s.v)
59     title("potential", fontsize=9)
60     tight_layout()
61
62
63     figure(figsize=(10, 1.75*nrows))
64     for j in 1:nplots-1
65         subplot(nrows, 5, j)
66         plot_complexvalued(o.x, u[:,(j-1)*thin+2], ymin, ymax)
67         title("j = $j", fontsize=9)
68     end
69     tight_layout()
70 end
71
72 function anim_Schroedinger(gifname, s::Schroedinger_Data, u, t; thin=2, interval=100, ymin=-1.1, ymax=1.1,
73 giftitle=gifname)
74     o = s.o
75     function plotframe(j)
        clf()

```

Out[17]: anim_Schroedinger (generic function with 1 method)

4.1 Gaussian packet

`f = GaussianPacket(x₀, p₀, σ, Δx)` は函数

$$f(x) = \frac{1}{(\pi\sigma^2)^{1/4}} \exp\left(\frac{ip_0(x - x_0)}{2}\right) \exp\left(-\frac{(x - x_0)^2}{2\sigma^2}\right)$$

になる。この函数の絶対値の2乗を \mathbb{R} 上で積分すると 1 になる。

我々は周期境界条件を扱っているので、本当はこれを周期的に足し合わせて周期函数化してから使うべきなのだが、面倒なのでそうしないことにする。

(f : GaussianPacket)(x) の形式でパラメータ x_0 , p_0 , σ , Δx を持つ函数を定義する方法については

- [Function-like objects \(https://docs.julialang.org/en/v1/manual/methods/index.html#Function-like-objects-1\)](https://docs.julialang.org/en/v1/manual/methods/index.html#Function-like-objects-1) (Julia 1.1 Documentation (<https://docs.julialang.org/en/v1/>))

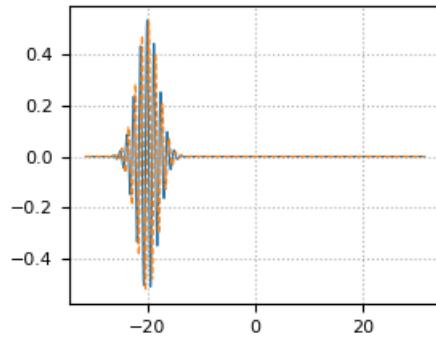
などを参照せよ。

In [18]:

```

1 ▼ mutable struct GaussianPacket{T<:Real}
2   x₀::T
3   p₀::T
4   σ::T
5   Δx::T
6 end
7
8 function GaussianPacket(x₀, p₀, σ, o::FFT_Data)
9   Δx = o.x[2] - o.x[1]
10  GaussianPacket(x₀, p₀, σ, Δx)
11 end
12
13 function (f::GaussianPacket)(x)
14   1/(π*f.σ^2)^(1/4) * exp(im*f.p₀*(x-f.x₀/2) - (x-f.x₀)^2/(2f.σ^2))
15 end
16
17 N = 2^10
18 K = 10
19 o = FFT_Data(K, N);
20
21 g₀ = GaussianPacket(-20.0, 5.0, 2.0, o)
22 plot_u(o.x, g₀.(o.x));

```



4.2 自由粒子

ポテンシャル函数がない場合の

$$i \frac{\partial}{\partial t} \psi = -\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 \psi$$

をGaussian packetの初期値で数値的に解いてみる。Gaussian packetが広がりながら波が進んで行く。

In [19]:

```

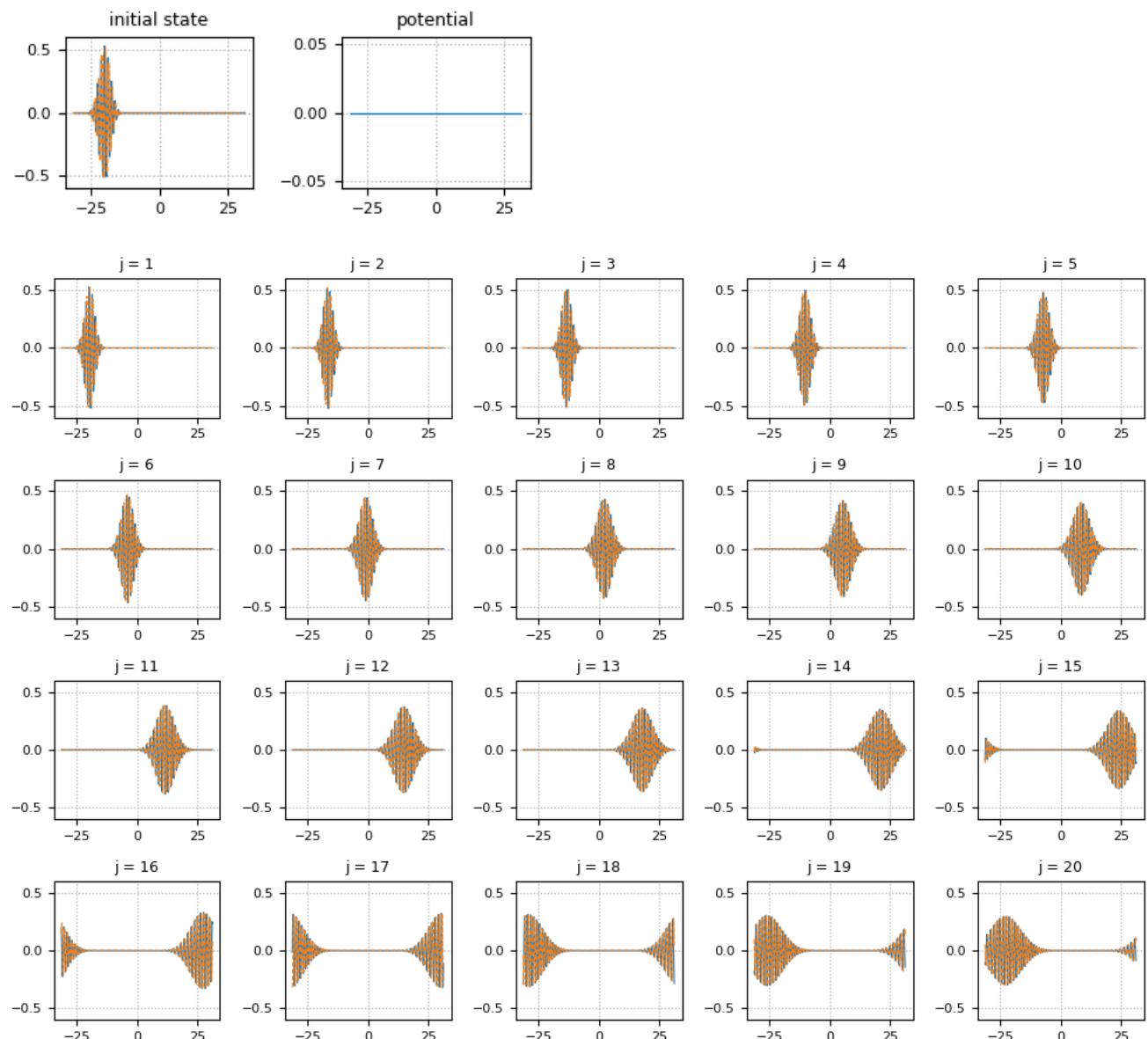
1 # 自由粒子
2
3 N = 2^10
4 K = 10
5 o = FFT_Data(K, N);
6 @show typeof(o)
7
8 V(x) = 0.0
9 s = Schroedinger_Data(o, V)
10
11 g0 = GaussianPacket(-20.0, 5.0, 2.0, o)
12 u0 = g0.(o.x)
13
14 T = 2.0
15 tmax = 2π*T
16 @time u, t = solve_Schroedinger(s, u0, tmax)
17
18 sleep(0.1)
19 plot_Schroedinger(s, u, t, ymin=-0.6, ymax=0.6)

```

```

typeof(o) = FFT_Data{Float64,FFTW.cFFTWPlan{Complex{Float64},-1,false,1}}
0.705572 seconds (1.40 M allocations: 74.363 MiB, 4.47% gc time)

```

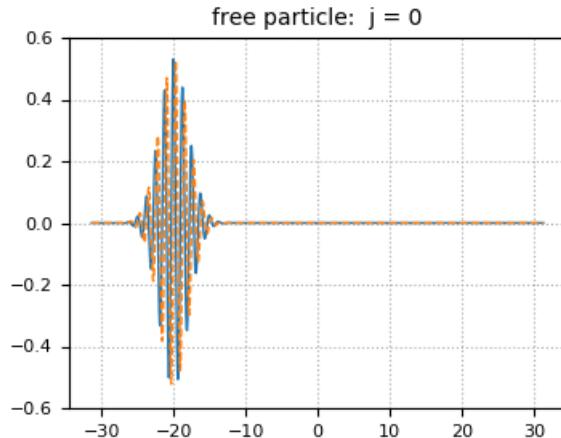


In [20]:

```

1 gifname = "free particle"
2 @time anim_Schroedinger(gifname, s, u, t, ymin=-0.6, ymax=0.6)

```



24.469933 seconds (363.78 k allocations: 35.937 MiB, 0.03% gc time)

4.3 低い壁

ポテンシャル函数が壁の形をした

$$V(x) = \begin{cases} 10 & (0 \leq x \leq 10) \\ 0 & (\text{otherwise}) \end{cases}$$

の場合の

$$i \frac{\partial}{\partial t} \psi = \left[-\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 + V(x) \right] \psi$$

を数値的に解いてみる。

低い壁に衝突した Gaussian packet の半分程度が壁を透過し、残りの半分程度が壁を反射している。

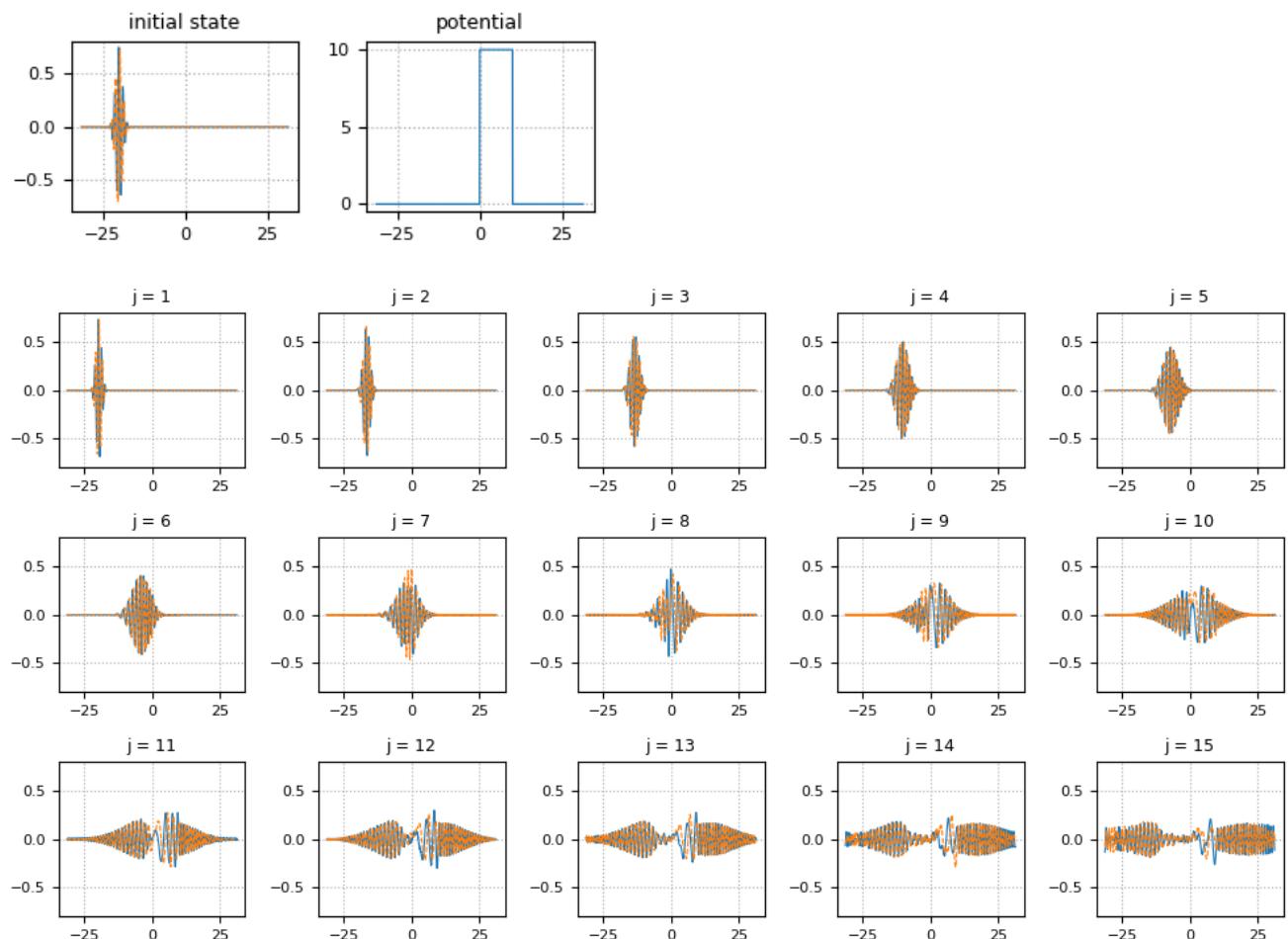
In [21]:

```

1 # 壁=(高さ10、幅10)、粒子=運動量5
2
3 N = 2^10
4 K = 10
5 o = FFT_Data(K, N);
6 @show typeof(o)
7
8 V(x) = ifelse(0 ≤ x ≤ 10, 10.0, 0.0)
9 s = Schroedinger_Data(o, V)
10
11 g0 = GaussianPacket(-20.0, 5.0, 1.0, o)
12 u0 = g0.(o.x)
13
14 T = 1.5
15 tmax = 2π*T
16 @time u, t = solve_Schroedinger(s, u0, tmax)
17
18 sleep(0.1)
19 plot_Schroedinger(s, u, t, ymin=-0.8, ymax=0.8)

```

`typeof(o) = FFT_Data{Float64,FFTW.cFFTWPlan{Complex{Float64},-1,false,1}}`
`0.269414 seconds (5.02 k allocations: 5.007 MiB)`

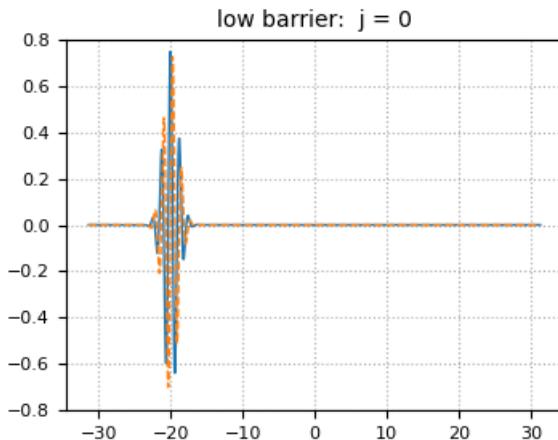


In [22]:

```

1 gifname = "low_barrier"
2 @time anim_Schroedinger(gifname, s, u, t, ymin=-0.8, ymax=0.8)

```



19.486258 seconds (65.82 k allocations: 14.667 MiB, 0.04% gc time)

4.4 高い壁

ポテンシャル函数が壁の形をした

$$V(x) = \begin{cases} 20 & (0 \leq x \leq 10) \\ 0 & (\text{otherwise}) \end{cases}$$

の場合の

$$i \frac{\partial}{\partial t} \psi = \left[-\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 + V(x) \right] \psi$$

を数値的に解いてみる。壁の高さは上の場合の2倍になっている。

高い壁に衝突した Gaussian packet の大部分が壁を反射している。

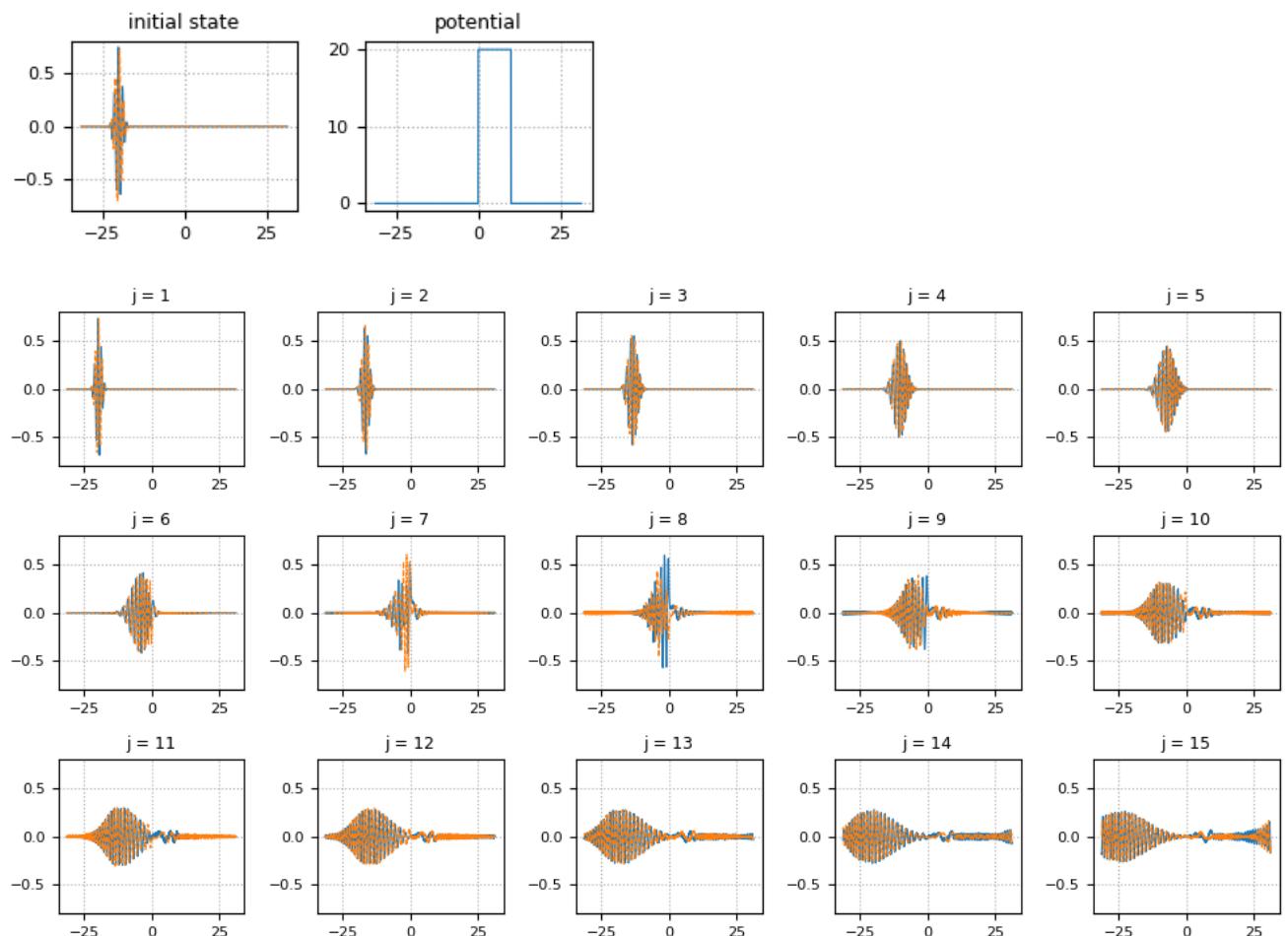
In [23]:

```

1 # 壁=(高さ20、幅10)、粒子=運動量5
2
3 N = 2^10
4 K = 10
5 o = FFT_Data(K, N);
6 @show typeof(o)
7
8 V(x) = ifelse(0 ≤ x ≤ 10, 20.0, 0.0)
9 s = Schroedinger_Data(o, V)
10
11 g0 = GaussianPacket(-20.0, 5.0, 1.0, o)
12 u0 = g0.(o.x)
13
14 T = 1.5
15 tmax = 2π*T
16 @time u, t = solve_Schroedinger(s, u0, tmax)
17
18 sleep(0.1)
19 plot_Schroedinger(s, u, t, ymin=-0.8, ymax=0.8)

```

`typeof(o) = FFT_Data{Float64,FFTW.cFFTWPlan{Complex{Float64},-1,false,1}}`
 0.321294 seconds (5.02 k allocations: 5.007 MiB)

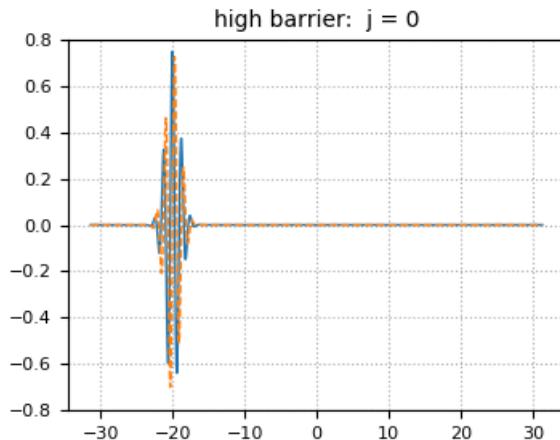


In [24]:

```

1 gifname = "high_barrier"
2 @time anim_Schroedinger(gifname, s, u, t, ymin=-0.8, ymax=0.8)

```



19.476868 seconds (65.27 k allocations: 14.494 MiB, 0.03% gc time)

4.5 調和振動子

時間に依存する量子調和振動子の方程式

$$i \frac{\partial}{\partial t} \psi = \left[-\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 + \frac{1}{2} x^2 \right] \psi$$

を数値的に解いてみる. Gaussian packet が壊れずに左右に振動する.

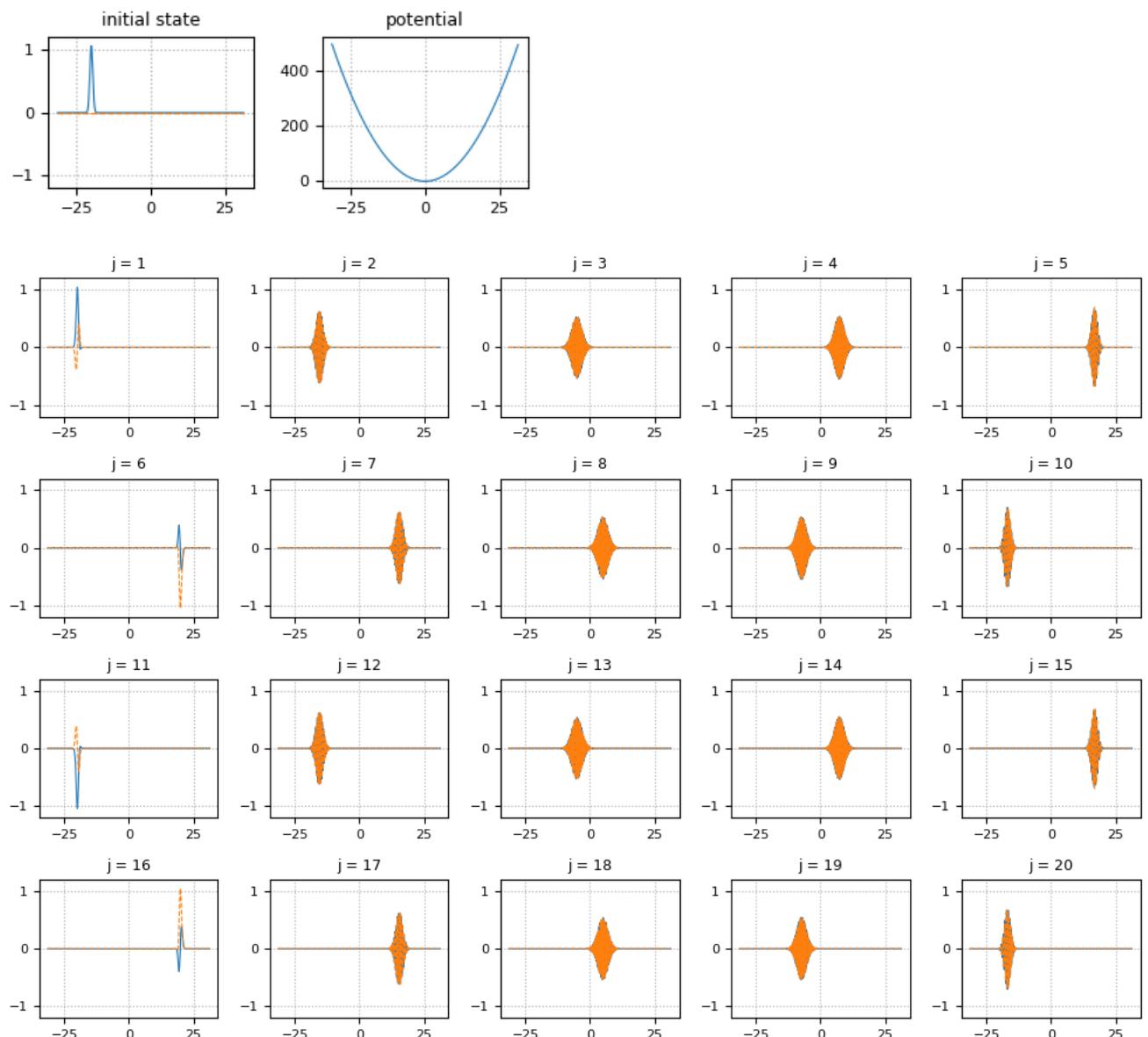
In [25]:

```

1 # 調和振動子 (周期 2π)
2
3 N = 2^10
4 K = 10
5 o = FFT_Data(K, N);
6 @show typeof(o)
7
8 V(x) = x^2/2
9 s = Schroedinger_Data(o, V)
10
11 g0 = GaussianPacket(-20.0, 0.0, 0.5, o)
12 u0 = g0.(o.x)
13
14 T = 2.0
15 tmax = 2π*T
16 @time u, t = solve_Schroedinger(s, u0, tmax)
17
18 sleep(0.1)
19 plot_Schroedinger(s, u, t, ymin=-1.2, ymax=1.2)

```

`typeof(o) = FFT_Data{Float64,FFTW.cFFTWPlan{Complex{Float64},-1,false,1}}`
`0.374075 seconds (6.68 k allocations: 6.664 MiB, 1.10% gc time)`

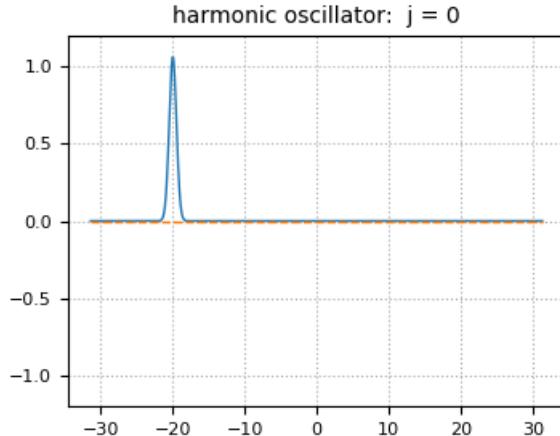


In [26]:

```

1 gifname = "harmonic oscillator"
2 @time anim_Schroedinger(gifname, s, u, t, ymin=-1.2, ymax=1.2, thin=1)

```



43.412640 seconds (154.40 k allocations: 30.416 MiB, 0.02% gc time)

4.6 $V(x) = -100 \exp(-x^2/100)$

ポテンシャルが

$$V(x) = -100 \exp(-x^2/100)$$

の場合の

$$i \frac{\partial}{\partial t} \psi = \left[-\frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 + V(x) \right] \psi$$

を数値的に解いてみる。

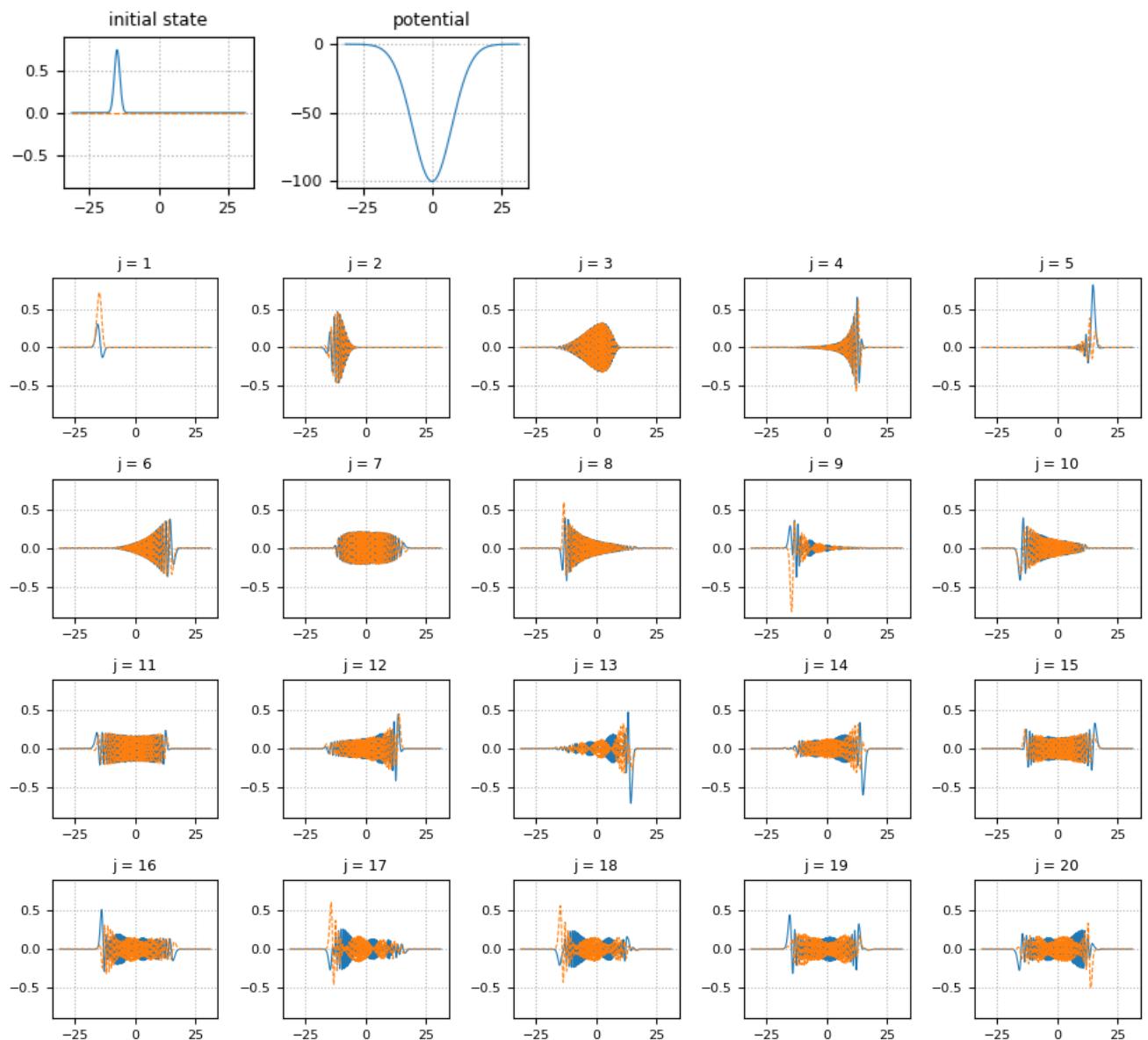
In [27]:

```

1 # V(x) = -20 exp(-x^2/25)
2
3 N = 2^10
4 K = 10
5 o = FFT_Data(K, N);
6 @show typeof(o)
7
8 V(x) = -100 * exp(-x^2/100)
9 s = Schroedinger_Data(o, V)
10
11 g0 = GaussianPacket(-15.0, 0.0, 1.0, o)
12 u0 = g0.(o.x)
13
14 T = 4.0
15 tmax = 2π*T
16 @time u, t = solve_Schroedinger(s, u0, tmax, skip=20)
17
18 sleep(0.1)
19 plot_Schroedinger(s, u, t, ymin=-0.9, ymax=0.9)

```

`typeof(o) = FFT_Data{Float64,FFTW.cFFTWPlan{Complex{Float64},-1,false,1}}`
 0.785552 seconds (110.36 k allocations: 12.037 MiB, 0.70% gc time)

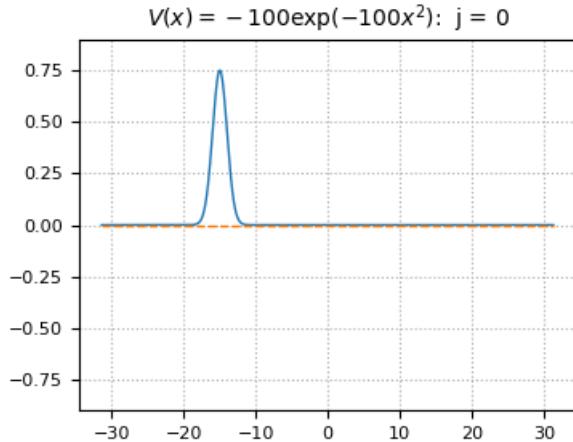


In [28]:

```

1 gifname = "V(x)=-100exp(-x2/100)"
2 giftitle = "\$V(x)=-100\exp(-100x^2)\$"
3 @time anim_Schroedinger(gifname, s, u, t, ymin=-0.9, ymax=0.9, giftitle=gifttitle)

```



26.148188 seconds (96.36 k allocations: 21.861 MiB, 0.02% gc time)

5 Smith方程式

KdV equation

$$u_t = -\partial_x^3 u - 3\partial_x(u^2).$$

に似ている

Smith's equation

$$u_t = 2a^{-2} \left(\sqrt{1 - a^2 \partial_x^2} - 1 \right) \partial_x u - 3\partial_x(u^2).$$

を数値的に解いてみる。以下では $a^2 = 0.15$ と仮定する。

Out[29]: anim Smith (generic function with 1 method)

```

1 function update_Smith!(o::FFT_Data, u1, u0, Δt, niters, a²)
2     v = similar(u0)
3     v_tmp = similar(u0)
4     u_tmp = similar(u0)
5
6     mul!(v, o.FFT, u0)
7     for iter in 1:niters
8         v .= exp.(2 ./ a² .* Δt .* (sqrt.(1 .- a² .* o.D2) .- 1).*o.D) .* v
9
10        # v .-= 3.0 .* Δt .* o.D .* (o.FFT * (o.FFT | v).^2)
11        ldiv!(u_tmp, o.FFT, v)
12        u_tmp .=* u_tmp
13        mul!(v_tmp, o.FFT, u_tmp)
14        v .-= 3.0 .* Δt .* o.D .* v_tmp
15    end
16    ldiv!(u1, o.FFT, v)
17 end
18
19 ▼ function solve_Smith_inplace(o::FFT_Data{T}, f0, tmax; a²=0.15) where T
20     Δt = 1/o.N^2
21     skip = floor(Int, 0.005/Δt)
22     t = 0:skip*Δt:tmax
23     M = length(t)
24     u = Array{Complex{T},2}(undef, o.N, M)
25     u[:,1] = Complex.(f0.(o.x))
26     for j in 2:M
27         update_Smith!(o, @view(u[:,j]), @view(u[:,j-1]), Δt, skip, a²)
28     end
29     return real(u), t
30 end

```

Out[30]: solve Smith inplace (generic function with 1 method)

5.1 Smith方程式: 初期条件 sin

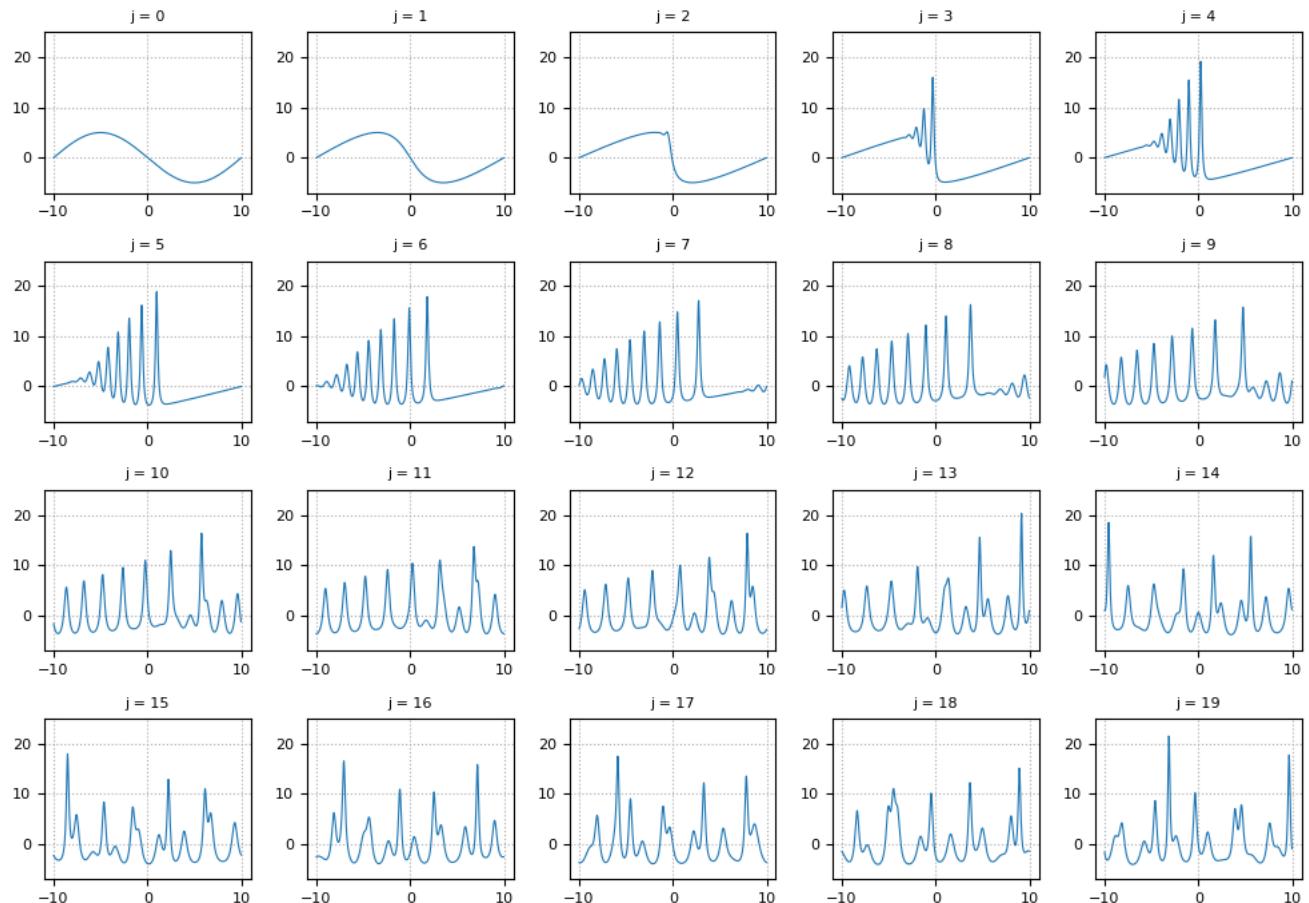
In [31]:

```

1 N = 2^9
2 K = 20/(2π)
3 o = FFT_Data(K, N);
4
5 f0(x) = -5*sin(π*x/10)
6 Δt = 1/N^2
7 skip = 10*floor(Int, 0.005/Δt)
8
9 tmax = 1.0
10 @time u, t = solve_Smith_inplace(o, f0, tmax)
11
12 sleep(0.1)
13 plot_Smith(o, u, t, thin=10)

```

36.299592 seconds (2.13 M allocations: 115.905 MiB, 0.08% gc time)

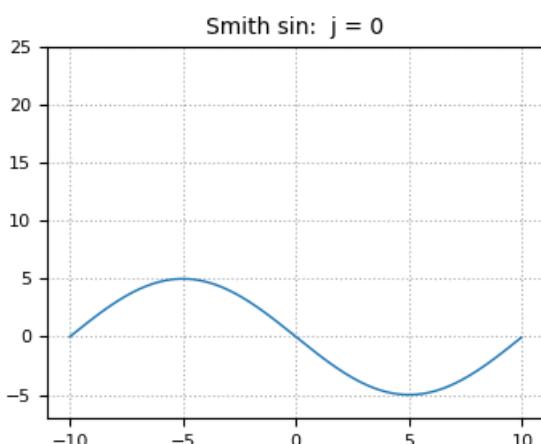


In [32]:

```

1 gifname = "Smith sin"
2 @time anim_Smith(gifname, o, u, t)

```



22.976449 seconds (314.46 k allocations: 22.931 MiB, 0.04% gc time)

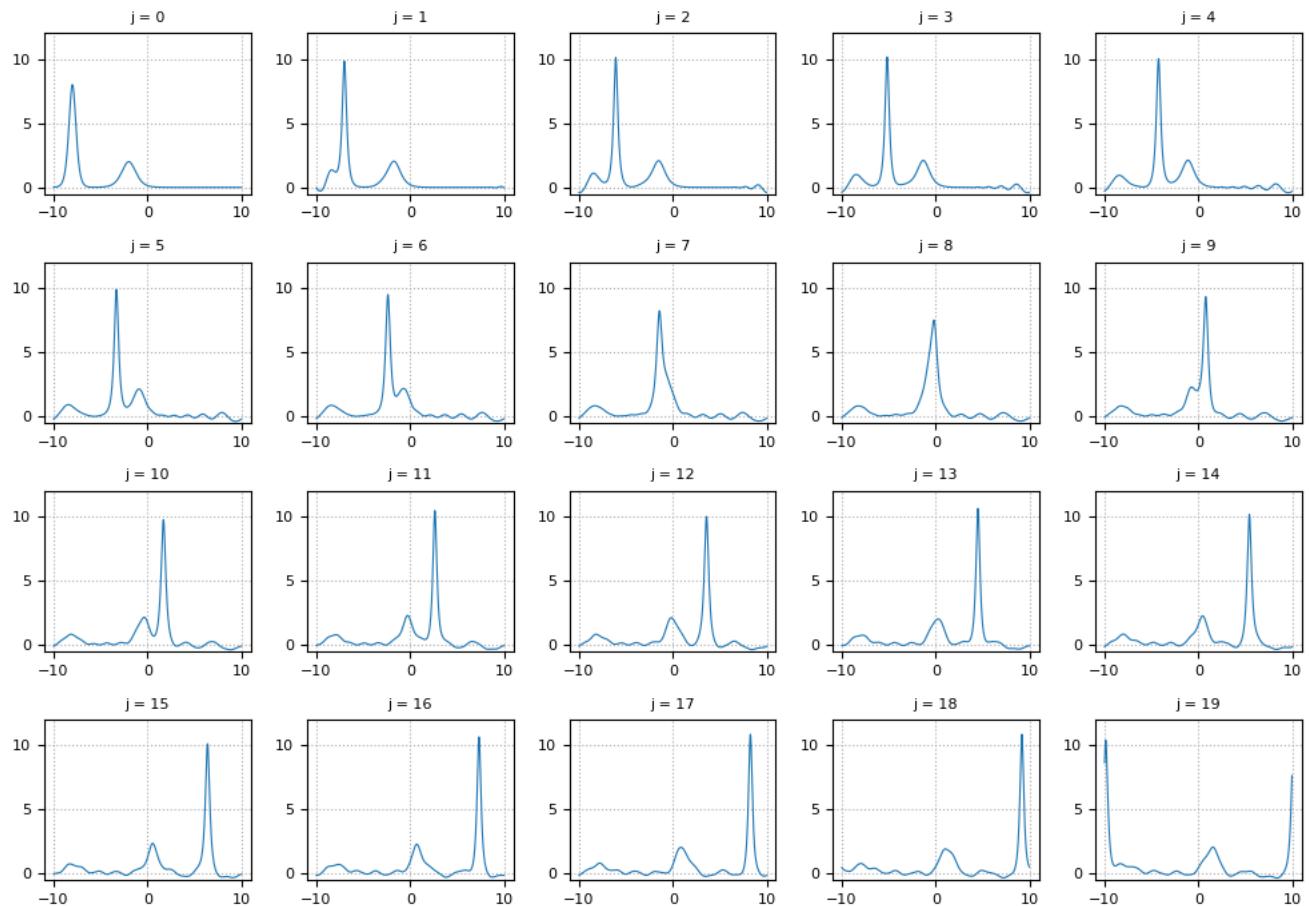
5.2 Smith方程式: 初期条件 KdV 2-soliton

```

In [33]: 1 N = 2^9
2 K = 20/(2π)
3 o = FFT_Data(K, N);
4
5 KdVsoltion(c, a, x) = c/2*(sech(sqrt(c)/2*(x-a)))^2
6 f0(x) = KdVsoltion(16.0, -8.0, x) + KdVsoltion(4.0, -2.0, x)
7 Δt = 1/N^2
8 skip = 10*floor(Int, 0.005/Δt)
9
10 tmax = 1.0
11 @time u, t = solve_Smith_inplace(o, f0, tmax)
12
13 sleep(0.1)
14 plot_Smith(o, u, t, ymin=-0.5, ymax=12.0)

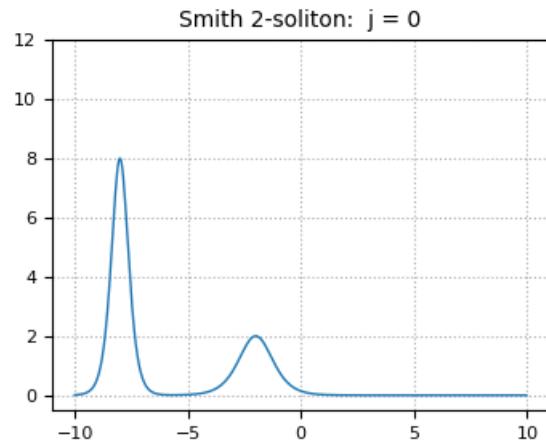
```

36.432395 seconds (912.03 k allocations: 57.313 MiB, 0.03% gc time)



In [34]:

```
1 gifname = "Smith 2-soliton"
2 @time anim_Smith(gifname, o, u, t, ymin=-0.5, ymax=12.0)
```



23.089005 seconds (73.48 k allocations: 10.103 MiB, 0.02% gc time)

古典型Cartan行列の対角化

- Author: 黒木玄
- Date: 2019-04-09～2019-04-21
- Copyright 2019 Gen Kuroki
- License: [The MIT License \(<https://opensource.org/licenses/MIT>\)](https://opensource.org/licenses/MIT)

古典型Cartan行列の固有ベクトルの表が検索しても容易に見付けることができなかつたので、その表を作ることにした。このノートでは、有限型の古典型Cartan行列とアフィン古典型の一般Cartan行列の固有ベクトルを求める。古典Cartan行列の特性多項式とChebyshev多項式の関係に関するよく知られた結果も解説する。

A_∞ 型Cartan行列は1次元格子 \mathbb{Z} 上の離散Laplacianであり、有限サイズの古典型Cartan行列は適当な境界条件を課すことによって得られる。

目次

1 古典型の場合

- 1.1 古典型Cartan行列と隣接行列の定義
- 1.2 計算の仕方の方針
- 1.3 無限隣接行列の定義
- 1.4 A_∞ 型
- 1.5 A_∞ 型Cartan行列と \mathbb{R} 上の正值Laplacianの関係
- 1.6 $A_{\infty/2}$ 型
- 1.7 A_n 型
- 1.8 C_∞ 型
- 1.9 C_n 型
- 1.10 B_∞ 型
- 1.11 B_n 型
- 1.12 D_∞ 型
- 1.13 D_{n+1} 型

2 古典アフィン型の場合

- 2.1 $A_{n-1}^{(1)}$ 型
- 2.2 $C_n^{(1)}$ 型
- 2.3 $A_{2n+2}^{(2)}$ 型
- 2.4 $D_{n+2}^{(2)}$ 型
- 2.5 $A_{2n+3}^{(2)}$ 型
- 2.6 $B_{n+1}^{(1)}$ 型
- 2.7 $D_{n+2}^{(1)}$ 型

3 Chebyshev多項式

- 3.1 Chebyshev多項式の定義
- 3.2 Chebyshev多項式の具体形
- 3.3 Chebyshev多項式の因数分解と隣接行列の特性多項式の関係
 - 3.3.1 第1種Chebyshev多項式の場合
 - 3.3.2 第2種Chebyshev多項式の場合
- 3.4 三角函数の無限積表示
 - 3.4.1 \cos の無限積表示
 - 3.4.2 \sin の無限積表示

In [1]:

```

1  using Plots
2  using LinearAlgebra
3  using SymPy: SymPy, sympy, @syms, @vars, simplify, PI, oo
4  const ChebyshevT = sympy.chebyshevT_poly
5  const ChebyshevU = sympy.chebyshevU_poly;

```

1 古典型の場合

1.1 古典型Cartan行列と隣接行列の定義

A_n, C_n, B_n, D_n 型のCartan行列とはそれぞれ以下の形の $n \times n$ 行列のことである.

A_n 型:

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

C_n 型:

$$\begin{bmatrix} 2 & -2 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

B_n 型:

$$\begin{bmatrix} 2 & -1 & & & \\ -2 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

D_n 型:

$$\begin{bmatrix} 2 & 0 & -1 & & \\ 0 & 2 & -1 & & \\ -1 & -1 & 2 & -1 & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

Cartan行列はどれも $2E - A$ (E は単位行列) の形をしている. そのとき A を隣接行列と呼ぶ. Cartan行列と隣接行列の固有ベクトルは一致し, Cartan行列の固有値は 2 から隣接行列の固有値を引いたものになる. したがって, Cartan行列の固有値と固有ベクトルを求めるためには, 対応する隣接行列のそれらを求めればよい.

上における C_n, B_n, D_n 型のCartan行列の表示において, 行列の成分の位置を表す番号 $1, 2, \dots, n$ を反転させればよく見る表示に戻る. 以下では1次元格子 \mathbb{Z} の 0 を境界条件を最初に設定する場所として選ぶので上のような表示を採用した.

1.2 計算の仕方の方針

我々は A_n, C_n, B_n, D_n 型の隣接行列の固有値と固有ベクトルを求めたい. しかし, 隣接行列の特性多項式を計算して, 固有値を計算して, 固有ベクトルを計算する経路を採用すると, 答えを知らないと非常に面倒な計算が必要になる.

そこで我々は, まず, 次の A_∞ 型の両無限隣接行列の固有ベクトルを構成し, 境界条件を設定した結果として, $A_{\infty/2}, C_\infty$ 型の半無限隣接行列の固有ベクトルを構成し, C_∞ 型の半無限隣接行列の固有ベクトルの最初の成分を半分にすることによって, B_∞ を重複させることによって, D_∞ 型の半無限隣接行列の固有ベクトルを構成する.

そして, 2つ目の境界条件を設定することによって, A_n, C_n, B_n, D_n 型隣接行列の固有ベクトルを得る.

1.3 無限隣接行列の定義

$A_\infty, A_{\infty/2}, C_\infty, B_\infty, D_\infty$ 型の隣接行列は以下のように定義される.

A_∞ 型の隣接行列とは次の $\mathbb{Z} \times \mathbb{Z}$ 行列のことである:

$$[\delta_{i,j-1} + \delta_{i,j-1}]_{i,j \in \mathbb{Z}} = \begin{bmatrix} \ddots & \ddots & & & \\ \ddots & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}.$$

$A_{\infty/2}$ 型の隣接行列とは次の $\{1, 2, \dots\} \times \{1, 2, \dots\}$ 行列のことである:

$$[\delta_{i,j-1} + \delta_{i,j-1}]_{i,j > 0} = \begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & \ddots \\ & & & 1 & 0 \\ & & & & \ddots & \ddots \end{bmatrix}.$$

C_∞ 型の隣接行列とは次の $\{0, 1, 2, \dots\} \times \{0, 1, 2, \dots\}$ 行列のことである:

$$\begin{bmatrix} 0 & 2 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & \ddots \\ & & & 1 & 0 \\ & & & & \ddots & \ddots \end{bmatrix}.$$

B_∞ 型の隣接行列とは次の $\{0, 1, 2, \dots\} \times \{0, 1, 2, \dots\}$ 行列のことである:

$$\begin{bmatrix} 0 & 1 & & & \\ 2 & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & \ddots \\ & & & 1 & 0 \\ & & & & \ddots & \ddots \end{bmatrix}.$$

D_∞ 型の隣接行列とは次の $\{0', 0, 1, 2, \dots\} \times \{0', 0, 1, 2, \dots\}$ 行列のことである:

$$\begin{bmatrix} 0 & 0 & 1 & & & \\ 0 & 0 & 1 & & & \\ 1 & 1 & 0 & 1 & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & \ddots & \\ & & & 1 & 0 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}.$$

1.4 A_∞ 型

A は A_∞ 型隣接行列($\mathbb{Z} \times \mathbb{Z}$ 行列)であるとする:

$$A = \begin{bmatrix} \ddots & \ddots & & & \\ \ddots & 0 & 1 & & \\ & 1 & 0 & 1 & \\ & & 1 & 0 & 1 \\ & & & 1 & 0 & \ddots \\ & & & & \ddots & \ddots \end{bmatrix}.$$

ベクトル $v = [x_j]_{j \in \mathbb{Z}} \neq 0$ が A の固有値 α の固有ベクトルであるための必要十分条件は

$$x_{j-1} + x_{j+1} = \alpha x_j \quad (j \in \mathbb{Z})$$

が成立することである。この定数係数の線形漸化式の解空間は常に2次元になるので、 A の固有空間の次元も常に2次元になる。

$\alpha \neq \pm 2$ のとき、 z に関する2次方程式 $z^2 - \alpha z - 1 = 0$ は異なる2つの解を持ち、その片方を z とするともう一方は z^{-1} になり、 $z \neq \pm 1$ となる。逆に、 $z \neq \pm 1$ のとき、 $\alpha = z + z^{-1}$ とおくと、 $\alpha \neq \pm 2$ となる。(例: $\alpha = 0$ のとき、 $z^{\pm 1} = \pm i$.)

ゆえに, A の固有値 $\alpha = z + z^{-1} \neq \pm 2$ の固有空間の基底として, $x_j = z^j, z^{-j}$ が取れる:

$$z^{j-1} + z^{j+1} = (z + z^{-1})z^j.$$

A の固有値 $\alpha = \pm 2$ の固有空間の基底として, $x_j = (\pm 1)^j, j(\pm 1)^{j-1}$ が取れる:

$$(j-1)(\pm 1)^{j-2} + (j+1)(\pm 1)^j = ((j-1)(\pm 1) + (j+1)(\pm 1))(\pm 1)^{j-1} = \pm 2j(\pm 1)^{j-1}.$$

1.5 A_∞ 型Cartan行列と \mathbb{R} 上の正値Laplacianの関係

\mathbb{R} 上の正値Laplacianとは $-(d/dt)^2$ のことである. $f(t)$ が C^2 級ならば

$$f(t \pm h) - f(t) = \pm f'(t)h + \frac{1}{2}f''(t)h^2 + o(h^2)$$

が成立している. ゆえに

$$-f(t-h) + 2f(t) - f(t+h) = -f''(t)h^2 + o(h^2)$$

なので

$$\lim_{h \rightarrow 0} \frac{-f(t-h) + 2f(t) - f(t+h)}{h^2} = -\left(\frac{d}{dt}\right)^2 f(t).$$

ゆえに, $f(t)$ の値を \mathbb{Z} 上に制限して得られる数列 $x_k = f(k) (k \in \mathbb{Z})$ を考えるとき, 無限次元ベクトル $[x_k]_{k \in \mathbb{Z}}$ を

$$[-x_{k-1} + 2x_k - x_{k+1}]_{k \in \mathbb{Z}}$$

に対応させる線形変換は正値Laplacianの離散化とみなされる. その線形変換を表現する行列は A_∞ 型Cartan行列に一致する.

$f(0) = 0$ という条件を課すことは, $x_0 = 0$ という条件を課すことに対応しており, そのとき A_∞ 型Cartan行列によって無限次元ベクトル $[x_k]_{k \in \mathbb{Z}}$ をうつすと, うつした先のベクトルの第1成分は

$$2x_1 - x_2$$

になり, x_0 が見掛け上見えなくなるので, A_∞ 型Cartan行列は $[x_k]_{k=1}^\infty$ にも自然に作用できるようになる. その表現行列が $A_{\infty/2}$ 型 Cartan行列になる. このように, 境界条件を設定することによって, A_∞ 型Cartan行列から, サイズの小さな別のCartan行列が得られる. 詳しくは以下の解説を参照せよ.

1.6 $A_{\infty/2}$ 型

A は A_∞ 型の両無限隣接行列であるとし, A_+ は $A_{\infty/2}$ 型の片無限隣接行列であるとする:

$$A = \begin{bmatrix} \ddots & \ddots & & & \\ \ddots & 0 & 1 & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & & \\ & & & 1 & 0 & \ddots & \\ & & & & \ddots & \ddots & \end{bmatrix}, \quad A_+ = \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & \ddots & \\ & & & \ddots & \ddots & \end{bmatrix}$$

ベクトル $v = [x_j]_{j \in \mathbb{Z}}$ が A の固有値 α の固有ベクトルのとき, $x_0 = 0$ ならば, ベクトル $v_+ = [x_j]_{j=1}^\infty$ は A_+ の固有値 α の固有ベクトルになる.

ゆえに, A_+ の固有値 $\alpha = z + z^{-1} \neq \pm 2 (z \neq \pm 1)$ の固有空間の基底として $[z^j - z^{-j}]_{j=0}^\infty$ が取れ, A_+ の固有値 $\alpha = \pm 2$ の固有空間の基底として $[j(\pm 1)^{j-1}]_{j=0}^\infty$ が取れる.

注意: 以上で課した x_j 達に関する条件 $x_0 = 0$ は奇函数の条件 $x_{-j} = -x_j$ と同値である. \square

1.7 A_n 型

A_n 型の隣接行列は次の形の $n \times n$ 行列になる.

$$A = \begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix}.$$

$A_{\infty/2}$ 型の隣接行列の固有値 $\alpha = z + z^{-1} \neq \pm 1$ の固有ベクトル $[z^k - z^{-k}]_{k=1}^{\infty}$, $z \neq \pm 1$ において, $z^{n+1} - z^{-(n+1)} = 1$ が成立しているとき, ゆえに特に $z = e^{i \cdot j\pi/(n+1)}$, $j = 1, 2, \dots, n$ のとき, ベクトル

$$[z^k - z^{-k}]_{k=1}^n = 2i \left[\sin \frac{kj\pi}{n+1} \right]_{k=1}^n$$

は A_n 型の隣接行列 A の固有値

$$z + z^{-1} = 2 \cos \frac{j\pi}{n+1}$$

の固有ベクトルになる. そこで θ_j を次のように定める:

$$\theta_j = \frac{j\pi}{n+1}.$$

このとき, A_n 型の隣接行列 A の互いに異なる n 個の固有値と対応する固有ベクトルとして以下が取れる:

$$\alpha_j = 2 \cos \theta_j, \quad v_j = \begin{bmatrix} \sin(1\theta_j) \\ \sin(2\theta_j) \\ \vdots \\ \sin(n\theta_j) \end{bmatrix} \quad (j = 1, 2, \dots, n).$$

以上の結果を直接確認したい場合には, $\sin 0\theta_j = \sin(n+1)\theta_j = 0$ と $\sin 2\theta_j = 2 \cos \theta_j \sin \theta_j$ および, $\sin((k \pm 1)\theta) = \cos \theta \sin(k\theta) \pm \sin \theta \cos(k\theta)$ より

$$\sin((k-1)\theta) + \sin((k+1)\theta) = 2 \cos \theta \sin(k\theta)$$

となることなどに注意せよ.

```
In [2]: 1 ▼ function adjacent_matrix_of_type_A(n)
2      @assert n ≥ 2
3      SymTridiagonal(zeros(Int,n), ones(Int,n-1))
4  end
5
6 ▼ function eigenvectors_of_type_A(n)
7      V = zeros(n, n)
8      for j in 1:n
9          θ_j = (j*π)/(n+1)
10     for i in 1:n
11         V[i,j] = 2sin(i*θ_j)
12     end
13 end
14 V
15 end
16
17 ▼ function eigenvalues_of_type_A(n)
18     α = zeros(n)
19     for j in 1:n
20         θ_j = j/(n+1)*π
21         α[j] = 2cos(θ_j)
22     end
23 α
24 end
```

Out[2]: eigenvalues_of_type_A (generic function with 1 method)

In [3]:

```

1 n = 11
2 A = adjacent_matrix_of_type_A(n)
3 V = eigenvectors_of_type_A(n)
4 α = eigenvalues_of_type_A(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

11x11 SymTridiagonal{Int64,Array{Int64,1}}:

```

0 1 . . . . . . .
1 0 1 . . . . . .
. 1 0 1 . . . . .
. . 1 0 1 . . . .
. . . 1 0 1 . . .
. . . . 1 0 1 . .
. . . . . 1 0 1 .
. . . . . . 1 0 1
. . . . . . . 1 0

```

11x11 Array{Float64,2}:

```

1.9 -0.0 0.0 0.0 0.0 0.0 -0.0 0.0 -0.0 0.0 -0.0
0.0 1.7 -0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.0
-0.0 0.0 1.4 0.0 0.0 0.0 0.0 0.0 0.0 -0.0 -0.0
-0.0 -0.0 0.0 1.0 0.0 -0.0 0.0 -0.0 0.0 0.0 0.0
-0.0 0.0 -0.0 0.0 0.5 0.0 -0.0 0.0 -0.0 -0.0 -0.0
0.0 -0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0
-0.0 0.0 -0.0 0.0 0.0 0.0 -0.5 0.0 -0.0 -0.0 -0.0
-0.0 0.0 0.0 -0.0 -0.0 0.0 0.0 -1.0 0.0 0.0 0.0
0.0 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 -1.4 -0.0 -0.0
-0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -1.7 0.0
0.0 -0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 -1.9

```

1x11 Array{Float64,2}:

```

1.93 1.73 1.41 1.0 0.52 0.0 -0.52 -1.0 -1.41 -1.73 -1.93

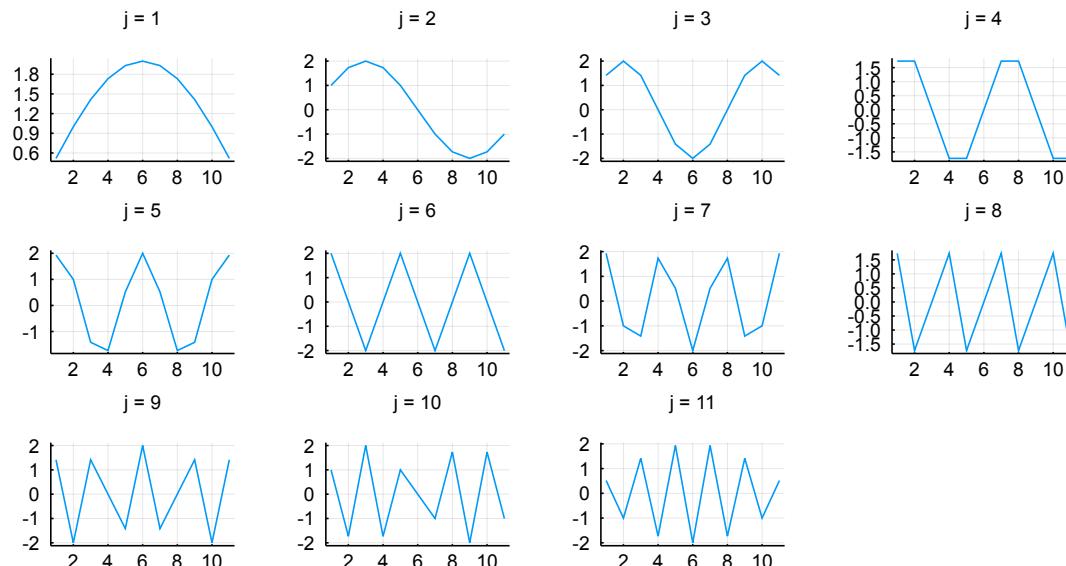
```

1x11 Array{Float64,2}:

```

1.93 1.73 1.41 1.0 0.52 0.0 -0.52 -1.0 -1.41 -1.73 -1.93

```



In [4]:

```

1 n = 12
2 A = adjacent_matrix_of_type_A(n)
3 V = eigenvectors_of_type_A(n)
4 α = eigenvalues_of_type_A(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

12×12 SymTridiagonal{Int64,Array{Int64,1}}:

```

0 1 . . . . . . . .
1 0 1 . . . . . . .
. 1 0 1 . . . . . .
. . 1 0 1 . . . . .
. . . 1 0 1 . . . .
. . . . 1 0 1 . . .
. . . . . 1 0 1 . .
. . . . . . 1 0 1 .
. . . . . . . 1 0 1
. . . . . . . . 1 0

```

12×12 Array{Float64,2}:

```

1.9 0.0 0.0 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0
0.0 1.8 -0.0 -0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
-0.0 0.0 1.5 0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.0
-0.0 -0.0 -0.0 1.1 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0 -0.0
-0.0 0.0 -0.0 0.0 0.7 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 0.0
0.0 -0.0 0.0 -0.0 -0.0 0.2 0.0 -0.0 -0.0 0.0 0.0 0.0 -0.0
0.0 0.0 0.0 0.0 0.0 -0.0 -0.2 0.0 -0.0 -0.0 -0.0 -0.0 0.0
0.0 -0.0 0.0 -0.0 0.0 0.0 0.0 -0.7 0.0 0.0 0.0 0.0 -0.0
-0.0 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.0 -1.1 -0.0 -0.0 0.0 0.0
0.0 -0.0 0.0 0.0 -0.0 0.0 -0.0 0.0 0.0 -1.5 0.0 0.0 0.0
-0.0 0.0 -0.0 -0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 -1.8 0.0
0.0 -0.0 0.0 0.0 0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 0.0 -1.9

```

1×12 Array{Float64,2}:

```

1.94 1.77 1.5 1.14 0.71 0.24 -0.24 -0.71 -1.14 -1.5 -1.77 -1.94

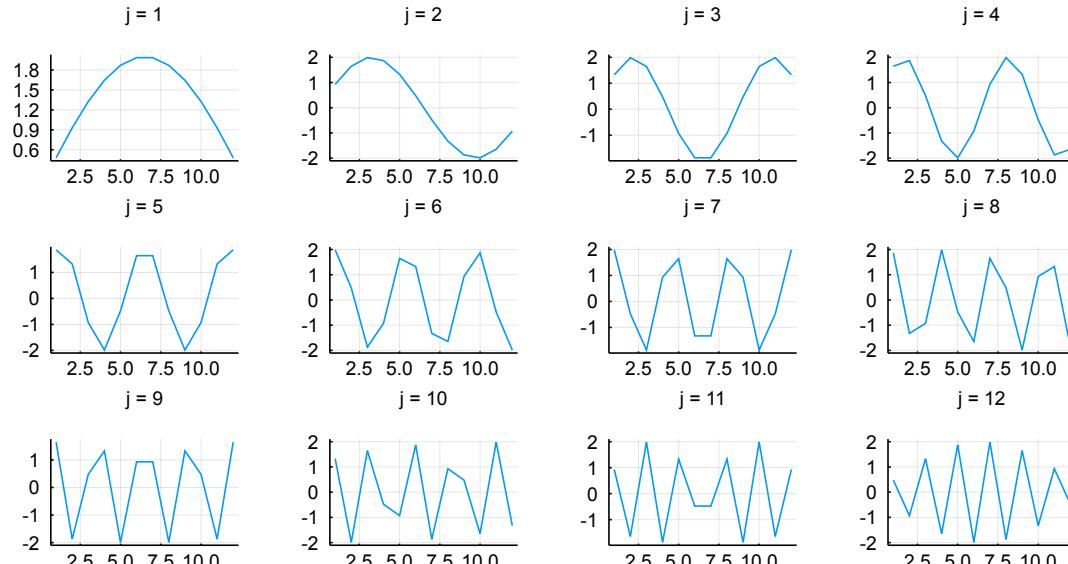
```

1×12 Array{Float64,2}:

```

1.94 1.77 1.5 1.14 0.71 0.24 -0.24 -0.71 -1.14 -1.5 -1.77 -1.94

```



1.8 C_∞ 型

ベクトル $v = [x_j]_{j \in \mathbb{Z}} \neq 0$ が A_∞ 型隣接行列の固有値 α の固有ベクトルになるための必要十分条件は

$$x_{j-1} + x_{j+1} = \alpha x_j$$

なので、もしも $x_{-j} = x_j$ が成立しているならば、

$$2x_1 = \alpha x_0$$

となるので、ベクトル $[x_j]_{j=0}^\infty$ は C_∞ 型隣接行列の固有値 α の固有ベクトルになる。

ゆえに、 C_∞ 型の隣接行列の固有値 $\alpha = z + z^{-1} \neq \pm 2$ ($z \neq \pm 1$) の固有空間の基底として $[z^j + z^{-j}]_{j=0}^\infty$ が取れ、固有値 $\alpha = \pm 2$ の固有空間の基底として $[(\pm 1)^j]_{j=0}^\infty$ が取れる。

1.9 C_n 型

C_n 型の隣接行列は次の形の $n \times n$ 行列になる：

$$A = \begin{bmatrix} 0 & 2 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix}.$$

以下ではこれを $\{0, 1, \dots, n-1\} \times \{0, 1, \dots, n-1\}$ 行列とみなす。

C_∞ 型の隣接行列の固有ベクトル $[z^k + z^{-k}]_{k=0}^\infty$ について、 $z^n + z^{-n} = 0$ が成立しているとき、ゆえに特に $z = e^{i \cdot (2j+1)\pi/(2n)}$, $j = 0, 1, \dots, n-1$ のとき、ベクトル

$$[z^k + z^{-k}]_{k=0}^{n-1} = 2 \left[\cos \frac{k(2j+1)\pi}{2n} \right]_{k=0}^{n-1}$$

は C_n 型隣接行列の固有値

$$z + z^{-1} = 2 \cos \frac{(2j+1)\pi}{2n}$$

の固有ベクトルである。そこで、 θ_j を次のように定める：

$$\theta_j = \frac{(2j+1)\pi}{2n}.$$

このとき、 C_n 型の隣接行列の互いに異なる n 個の固有値と対応する固有ベクトルとして以下が取れる：

$$\alpha_j = 2 \cos \theta_j, \quad v_j = \begin{bmatrix} \cos(0\theta_j) \\ \cos(1\theta_j) \\ \cos(2\theta_j) \\ \vdots \\ \cos((n-1)\theta_j) \end{bmatrix} \quad (j = 0, 1, \dots, n-1).$$

以上の結果を直接確認したい場合には、 $\cos((k \pm 1)\theta) = \cos \theta \cos(k\theta) \mp \sin \theta \sin(k\theta)$ より

$$\cos((k-1)\theta) + \cos((k+1)\theta) = 2 \cos \theta \cos(k\theta)$$

となることなどに注意せよ。

In [5]:

```
1 ▼ function adjacent_matrix_of_type_C(n)
2     @assert n ≥ 2
3     Tridiagonal(ones(Int,n-1), zeros(Int,n), [2; ones(Int,n-2)])
4 end
5
6 ▼ function eigenvectors_of_type_C(n)
7     V = zeros(n, n)
8     for j in 1:n
9         θ_j = (2j-1)*π/(2n)
10    for i in 1:n
11        V[i,j] = cos((i-1)*θ_j)
12    end
13 end
14 V
15 end
16
17 ▼ function eigenvalues_of_type_C(n)
18     α = zeros(n)
19     for j in 1:n
20         θ_j = (2j-1)/n*π/2
21         α[j] = 2cos(θ_j)
22     end
23 α
24 end
```

Out[5]: eigenvectors_of_type_C (generic function with 1 method)

In [6]:

```

1 n = 11
2 A = adjacent_matrix_of_type_C(n)
3 V = eigenvectors_of_type_C(n)
4 α = eigenvalues_of_type_C(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

11x11 Tridiagonal{Int64,Array{Int64,1}}:

```

0 2 . . . . . . .
1 0 1 . . . . . .
. 1 0 1 . . . . .
. . 1 0 1 . . . .
. . . 1 0 1 . . .
. . . . 1 0 1 . .
. . . . . 1 0 1 .
. . . . . . 1 0 1
. . . . . . . 1 0

```

11x11 Array{Float64,2}:

```

2.0 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.0 -0.0 0.0 0.0
0.0 1.8 0.0 -0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 -0.0
0.0 0.0 1.5 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -0.0 -0.0 1.1 0.0 -0.0 -0.0 -0.0 0.0 -0.0 0.0
0.0 -0.0 -0.0 0.0 0.6 0.0 0.0 0.0 -0.0 0.0 0.0
-0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 0.0 -0.0 0.0
-0.0 0.0 0.0 0.0 -0.0 0.0 -0.6 0.0 -0.0 0.0 -0.0
-0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -1.1 0.0 -0.0 0.0
0.0 -0.0 -0.0 0.0 0.0 0.0 0.0 0.0 -1.5 0.0 0.0
-0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.0 -1.8 -0.0
0.0 0.0 -0.0 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 -2.0

```

1x11 Array{Float64,2}:

```

1.98 1.82 1.51 1.08 0.56 -0.0 -0.56 -1.08 -1.51 -1.82 -1.98

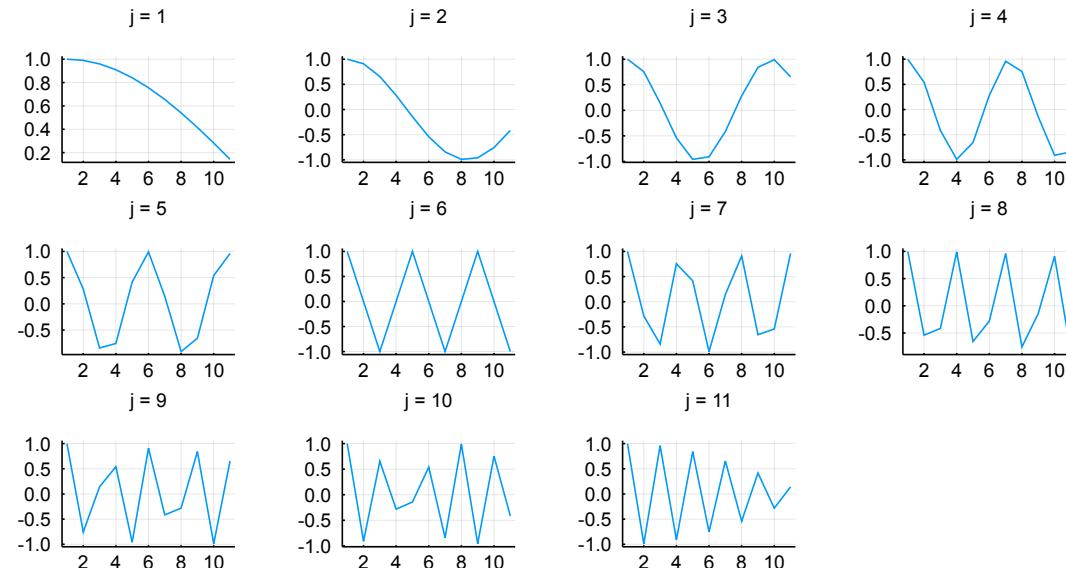
```

1x11 Array{Float64,2}:

```

1.98 1.82 1.51 1.08 0.56 0.0 -0.56 -1.08 -1.51 -1.82 -1.98

```



In [7]:

```

1 n = 12
2 A = adjacent_matrix_of_type_C(n)
3 V = eigenvectors_of_type_C(n)
4 α = eigenvalues_of_type_C(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

12×12 Tridiagonal{Int64,Array{Int64,1}}:

```

0 2 . . . . . . . .
1 0 1 . . . . . . .
. 1 0 1 . . . . . .
. . 1 0 1 . . . . .
. . . 1 0 1 . . . .
. . . . 1 0 1 . . .
. . . . . 1 0 1 . .
. . . . . . 1 0 1 .
. . . . . . . 1 0 1
. . . . . . . . 1 0

```

12×12 Array{Float64,2}:

```

2.0 -0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0
-0.0 1.8 0.0 -0.0 0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0 0.0 0.0
-0.0 -0.0 1.6 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 0.0 0.0 0.0 -0.0
0.0 -0.0 -0.0 1.2 0.0 -0.0 0.0 -0.0 0.0 -0.0 0.0 0.0 0.0
-0.0 0.0 0.0 0.0 0.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 -0.0 -0.0 0.0 0.0 0.3 0.0 -0.0 -0.0 -0.0 0.0 0.0 0.0
0.0 0.0 0.0 -0.0 -0.0 0.0 -0.3 0.0 0.0 0.0 0.0 -0.0 -0.0
0.0 -0.0 -0.0 0.0 0.0 -0.0 0.0 -0.8 -0.0 -0.0 0.0 0.0 -0.0
0.0 0.0 0.0 0.0 -0.0 0.0 -0.0 0.0 -1.2 0.0 -0.0 0.0 0.0
-0.0 -0.0 -0.0 0.0 0.0 0.0 -0.0 -0.0 -0.0 -0.0 -1.6 0.0 -0.0
0.0 0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0 -0.0 -1.8 0.0
-0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -0.0 -2.0

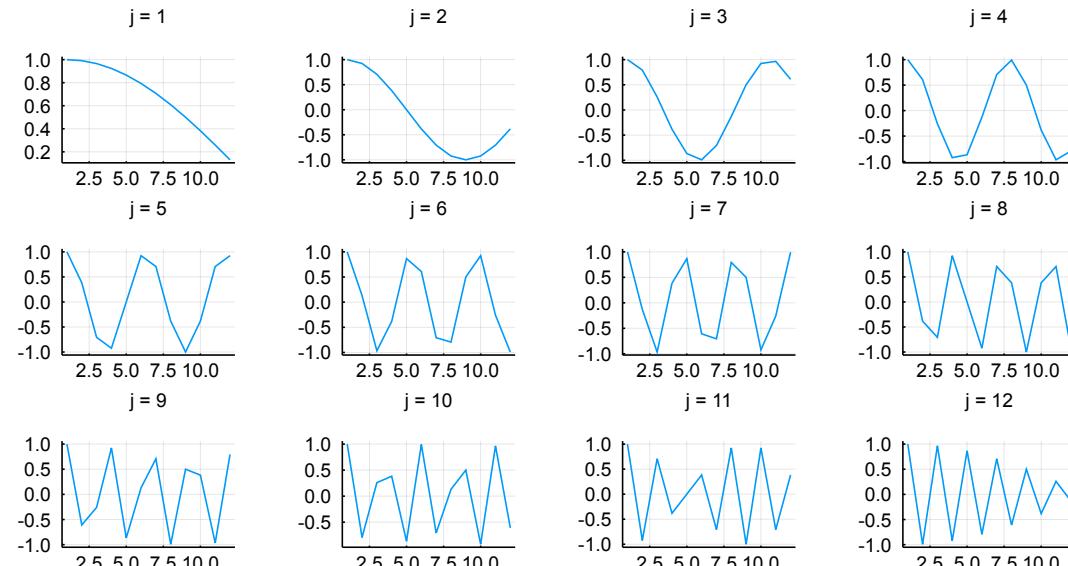
```

1×12 Array{Float64,2}:

```
1.98 1.85 1.59 1.22 0.77 0.26 -0.26 -0.77 -1.22 -1.59 -1.85 -1.98
```

1×12 Array{Float64,2}:

```
1.98 1.85 1.59 1.22 0.77 0.26 -0.26 -0.77 -1.22 -1.59 -1.85 -1.98
```



1.10 B_∞ 型

ベクトル $v = [x_k]_{k=0}^{\infty} \neq 0$ が C_{∞} 型隣接行列の固有値 α の固有ベクトルであることの必要十分条件は

$$2x_1 = \alpha x_0, x_0 + x_2 = \alpha x_1, x_1 + x_3 = \alpha x_2, x_2 + x_4 = \alpha x_3, \dots$$

が成立することであり、これは

$$x_1 = \alpha \frac{x_0}{2}, 2 \frac{x_0}{2} + x_2 = \alpha x_1, x_1 + x_3 = \alpha x_2, x_2 + x_4 = \alpha x_3, \dots$$

と同値であり、 v の最初の成分を半分にしたもののは B_n 型隣接行列の固有値になっていることがわかる。

ゆえに、 B_{∞} 型の隣接行列の固有値 $\alpha = z + z^{-1} \neq \pm 2$ ($z \neq \pm 1$) の固有空間の基底として

$$\begin{bmatrix} 1 \\ z + z^{-1} \\ z^2 + z^{-2} \\ z^3 + z^{-3} \\ \vdots \end{bmatrix}$$

が取れ、固有値 $\alpha = \pm 2$ の固有空間の基底として

$$\begin{bmatrix} 1/2 \\ -1 \\ 1 \\ -1 \\ 1 \\ \vdots \end{bmatrix}$$

が取れる。

1.11 B_n 型

B_n 型の隣接行列は次の形の $n \times n$ 行列になる:

$$A = \begin{bmatrix} 0 & 1 & & & \\ 2 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix}.$$

以下ではこれを $\{0, 1, \dots, n-1\} \times \{0, 1, \dots, n-1\}$ 行列とみなす。

θ_j を次のように定める:

$$\theta_j = \frac{(2j+1)\pi}{2n}.$$

前節の結果を使うと、 B_n 型の隣接行列の互いに異なる n 個の固有値と対応する固有ベクトルとして以下が取れることがわかる:

$$\alpha_j = 2 \cos \theta_j, \quad v_j = \begin{bmatrix} 1/2 \\ \cos(1\theta_j) \\ \cos(2\theta_j) \\ \vdots \\ \cos((n-1)\theta_j) \end{bmatrix} \quad (j = 0, 1, \dots, n-1).$$

この固有ベクトルは C_n 型の場合の固有ベクトルの第1成分を半分にしたものになっている。 $v = [x_k]_{k=0}^{n-1} \neq 0$ が C_n 型の隣接行列の固有値 α の固有ベクトルであることの必要十分条件は、

$$2x_1 = \alpha x_0, x_0 + x_2 = \alpha x_1, \dots, x_{n-3} + x_{n-1} = \alpha x_{n-2}, x_{n-2} = \alpha x_{n-1}$$

が成立することであり、これは

$$x_1 = \alpha \frac{x_0}{2}, 2 \frac{x_0}{2} + x_2 = \alpha x_1, \dots, x_{n-3} + x_{n-1} = \alpha x_{n-2}, x_{n-2} = \alpha x_{n-1}$$

と同値であるので、 v の最初の成分を半分にしたもののは B_n 型の隣接行列の固有ベクトルになっていることがわかる。

In [8]:

```
1 ▼ function adjacent_matrix_of_type_B(n)
2     @assert n ≥ 2
3     Tridiagonal([2; ones(Int,n-2)], zeros(Int,n), ones(Int,n-1))
4 end
5
6 ▼ function eigenvectors_of_type_B(n)
7     V = zeros(n, n)
8     for j in 1:n
9         θ_j = (2j-1)*π/(2n)
10        V[1,j] = 1/2
11        for i in 2:n
12            V[i,j] = cos((i-1)*θ_j)
13        end
14    end
15    V
16 end
17
18 ▼ function eigenvalues_of_type_B(n)
19     α = zeros(n)
20     for j in 1:n
21         θ_j = (2j-1)/n*π/2
22         α[j] = 2cos(θ_j)
23     end
24     α
25 end
```

Out[8]: eigenvectors_of_type_B (generic function with 1 method)

In [9]:

```

1 n = 11
2 A = adjacent_matrix_of_type_B(n)
3 V = eigenvectors_of_type_B(n)
4 α = eigenvalues_of_type_B(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

11x11 Tridiagonal{Int64,Array{Int64,1}}:

```

0 1 . . . . . . .
2 0 1 . . . . . . .
. 1 0 1 . . . . .
. . 1 0 1 . . . .
. . . 1 0 1 . . .
. . . . 1 0 1 . .
. . . . . 1 0 1 .
. . . . . . 1 0 1
. . . . . . . 1 0

```

11x11 Array{Float64,2}:

```

2.0 -0.0 0.0 0.0 0.0 -0.0 0.0 0.0 -0.0 0.0 0.0
0.0 1.8 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 -0.0
0.0 0.0 1.5 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 0.0
-0.0 -0.0 0.0 1.1 0.0 -0.0 -0.0 -0.0 0.0 -0.0 0.0
-0.0 0.0 -0.0 0.0 0.6 0.0 0.0 0.0 -0.0 0.0 0.0
-0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -0.0 0.0 -0.0 0.0
-0.0 0.0 0.0 0.0 -0.0 0.0 -0.6 0.0 -0.0 0.0 -0.0
-0.0 -0.0 -0.0 -0.0 0.0 -0.0 -0.0 -1.1 0.0 -0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.5 0.0 0.0
-0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -1.8 0.0
0.0 0.0 -0.0 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 -2.0

```

1x11 Array{Float64,2}:

```

1.98 1.82 1.51 1.08 0.56 -0.0 -0.56 -1.08 -1.51 -1.82 -1.98

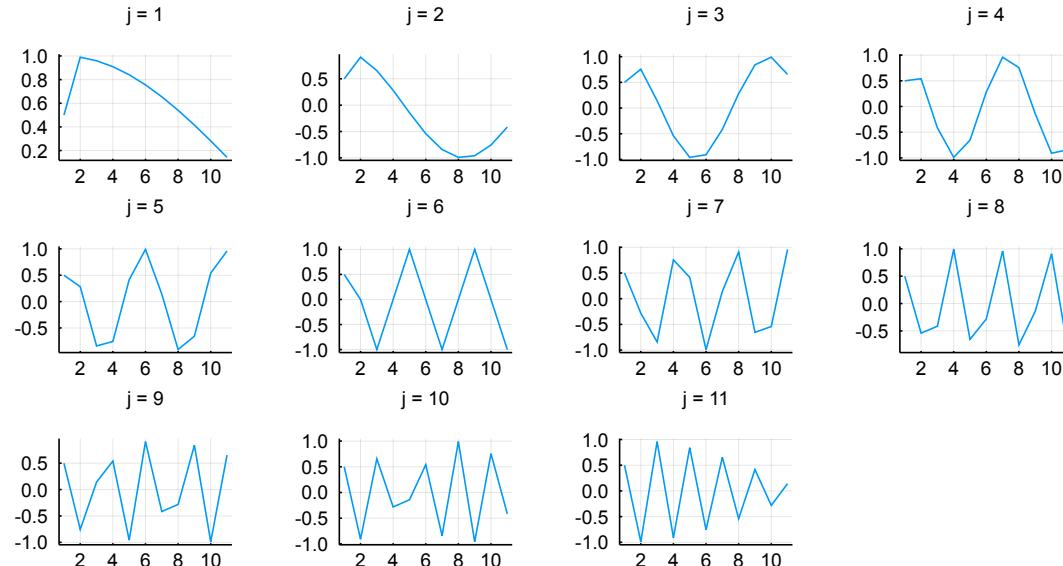
```

1x11 Array{Float64,2}:

```

1.98 1.82 1.51 1.08 0.56 0.0 -0.56 -1.08 -1.51 -1.82 -1.98

```



In [10]:

```

1 n = 12
2 A = adjacent_matrix_of_type_B(n)
3 V = eigenvectors_of_type_B(n)
4 α = eigenvalues_of_type_B(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

12×12 Tridiagonal{Int64,Array{Int64,1}}:

```

0 1 . . . . . . . .
2 0 1 . . . . . . . .
. 1 0 1 . . . . . .
. 1 0 1 . . . . . .
. . 1 0 1 . . . . .
. . . 1 0 1 . . . .
. . . . 1 0 1 . . .
. . . . . 1 0 1 . .
. . . . . . 1 0 1 .
. . . . . . . 1 0 1
. . . . . . . . 1 0

```

12×12 Array{Float64,2}:

```

2.0 -0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0 0.0 -0.0
-0.0 1.8 -0.0 -0.0 0.0 -0.0 0.0 -0.0 0.0 -0.0 -0.0 -0.0
-0.0 0.0 1.6 -0.0 -0.0 0.0 -0.0 0.0 -0.0 0.0 0.0 -0.0
0.0 -0.0 0.0 1.2 0.0 -0.0 0.0 -0.0 0.0 -0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.8 0.0 0.0 0.0 -0.0 0.0 0.0 -0.0
0.0 -0.0 -0.0 0.0 0.0 0.3 0.0 -0.0 -0.0 -0.0 0.0 0.0
0.0 0.0 0.0 -0.0 -0.0 0.0 -0.3 0.0 -0.0 0.0 -0.0 0.0
0.0 -0.0 -0.0 0.0 0.0 -0.0 -0.0 -0.8 -0.0 -0.0 -0.0 -0.0
0.0 0.0 0.0 -0.0 -0.0 0.0 -0.0 0.0 -1.2 0.0 -0.0 0.0
-0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 -0.0 -0.0 -0.0 -1.6 0.0 -0.0
0.0 0.0 0.0 0.0 0.0 -0.0 0.0 0.0 -0.0 0.0 -1.8 -0.0
-0.0 -0.0 0.0 -0.0 -0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 -2.0

```

1×12 Array{Float64,2}:

```

1.98 1.85 1.59 1.22 0.77 0.26 -0.26 -0.77 -1.22 -1.59 -1.85 -1.98

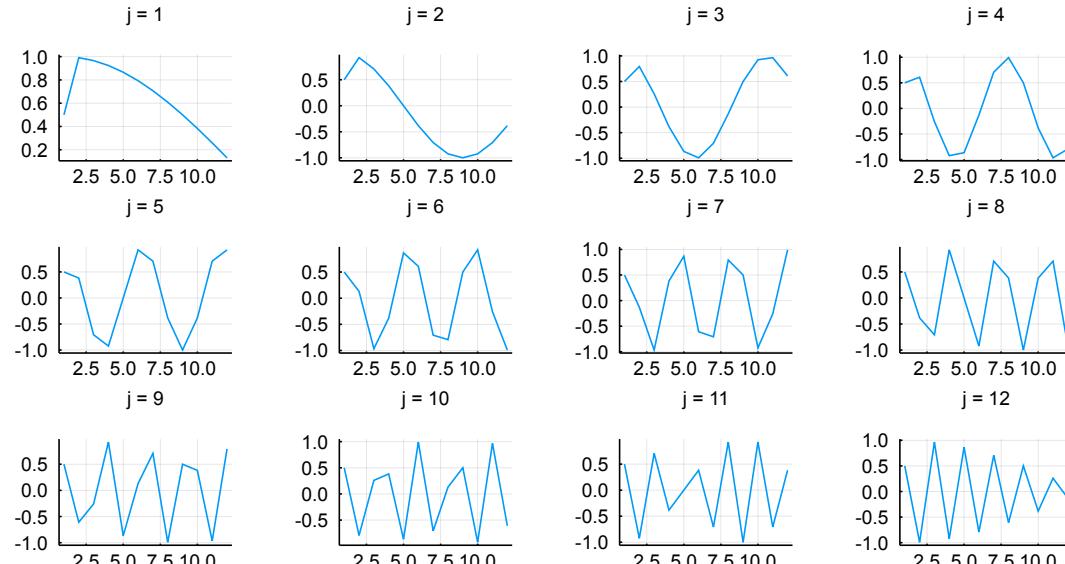
```

1×12 Array{Float64,2}:

```

1.98 1.85 1.59 1.22 0.77 0.26 -0.26 -0.77 -1.22 -1.59 -1.85 -1.98

```



1.12 D_∞ 型

ベクトル $v = [x_k]_{k=0',0,1,2,\dots} \neq 0$ が D_∞ 型隣接行列の固有値 α の固有ベクトルになるための必要十分条件は

$$x_1 = \alpha x_{0'}, x_1 = \alpha x_0, x_{0'} + x_0 + x_2 = \alpha x_1, x_1 + x_3 = \alpha x_2, x_2 + x_4 = \alpha x_3, \dots$$

が成立することである。一方、ベクトル $w = [y_k]_{k=0}^\infty \neq 0$ が B_∞ 型隣接行列の固有値 α の固有ベクトルになるための必要十分条件は、

$$y_1 = \alpha y_0, 2y_0 + y_2 = \alpha y_1, y_1 + y_3 = \alpha y_2, y_2 + y_4 = \alpha y_3, \dots$$

が成立することである。 B_∞ 型隣接行列の固有値 α の固有ベクトル $w = [y_k]_{k=0}^\infty \neq 0$ に対して、 $x_{0'} = y_0, x_k = y_k$ とおくと、ベクトル $v = [x_k]_{k=0',0,1,2,\dots} \neq 0$ は D_∞ 型隣接行列の固有値 α の固有ベクトルになる。

$$x_1 = \alpha \frac{x_0}{2}, 2\frac{x_0}{2} + x_2 = \alpha x_1, x_1 + x_3 = \alpha x_2, x_2 + x_4 = \alpha x_3, \dots$$

と同値であり、 v の最初の成分を半分にしたものは B_n 型隣接行列の固有値になっていることがわかる。

ゆえに、 D_∞ 型の隣接行列の固有値 $\alpha = z + z^{-1} \neq \pm 2$ ($z \neq \pm 1$) の固有空間の基底として

$$\begin{bmatrix} 1 \\ 1 \\ z + z^{-1} \\ z^2 + z^{-2} \\ z^3 + z^{-3} \\ \vdots \end{bmatrix}$$

が取れ、固有値 $\alpha = \pm 2$ の固有空間の基底として

$$\begin{bmatrix} 1/2 \\ 1/2 \\ -1 \\ 1 \\ -1 \\ 1 \\ \vdots \end{bmatrix}$$

が取れる。

以上のようにして作られた D_∞ 型隣接行列の固有ベクトルでは最初の2つの成分が等しくなる。そうではない固有値 0 の固有ベクトルを

$$v = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} = [\delta_{k,0'} - \delta_{k,0}]_{k=0',0,1,2,\dots}$$

によって作ることができる。

1.13 D_{n+1} 型

次の形の $(n+1) \times (n+1)$ 行列を D_{n+1} 型の隣接行列と呼ぶ:

$$A = \begin{bmatrix} 0 & 0 & 1 & & & & \\ 0 & 0 & 1 & & & & \\ 1 & 1 & 0 & 1 & & & \\ & & 1 & 0 & \ddots & & \\ & & & \ddots & \ddots & 1 & \\ & & & & 1 & 0 & \end{bmatrix}.$$

θ_j を次のように定める:

$$\theta_j = \frac{(2j+1)\pi}{2n}.$$

このとき、前節の結果より、 D_{n+1} 型の隣接行列の固有値と対応する固有ベクトルとして以下が取れることがわかる:

$$\alpha_j = 2 \cos \theta_j, \quad v_j = \begin{bmatrix} 1/2 \\ 1/2 \\ \cos(1\theta_j) \\ \cos(2\theta_j) \\ \vdots \\ \cos((n-1)\theta_j) \end{bmatrix} \quad (j = 0, 1, \dots, n-1),$$

$$\alpha_n = 0, \quad v_n = \begin{bmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

v_0, v_1, \dots, v_{n-1} は B_n 型隣接行列の固有ベクトルの第1成分を重複させたものに等しい。そのことから v_0, v_1, \dots, v_{n-1} が D_{n+1} 型隣接行列の固有ベクトルであることがわかる。 v_n が D_{n+1} 型隣接行列の固有値 0 の固有ベクトルであることは自明である。

$n+1$ が偶数のとき, 0 は D_{n+1} 型隣接行列の固有値として2重に重複している。 $n+1$ が奇数のとき, $n+1$ 個の固有値 α_j は互いに異なる。

In [11]:

```

1 ▼ function adjacent_matrix_of_type_D(n)
2     @assert n ≥ 3
3     G = zeros(Int, n, n)
4     G[1,3] = 1
5 ▼     for i in 2:n-1
6         G[i,i+1] = 1
7     end
8     Symmetric(G)
9 end
10
11 ▼ function eigenvectors_of_type_D(n)
12     V = zeros(n, n)
13 ▼     for j in 1:n-1
14         θ_j = (2j-1)*π/(2*(n-1))
15         V[1,j] = V[2,j] = 1/2
16 ▼         for i in 3:n
17             V[i,j] = cos((i-2)*θ_j)
18         end
19     end
20     V[1,n] = 1
21     V[2,n] = -1
22     V
23 end
24
25 ▼ function eigenvalues_of_type_D(n)
26     α = zeros(n)
27 ▼     for j in 1:n-1
28         θ_j = (2j-1)/(n-1)*π/2
29         α[j] = 2cos(θ_j)
30     end
31     α[n] = 0
32     α
33 end

```

Out[11]: eigenvectors_of_type_D (generic function with 1 method)

In [12]:

```

1 n = 11
2 A = adjacent_matrix_of_type_D(n)
3 V = eigenvectors_of_type_D(n)
4 α = eigenvalues_of_type_D(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

11x11 Symmetric{Int64,Array{Int64,2}}:

0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	0

11x11 Array{Float64,2}:

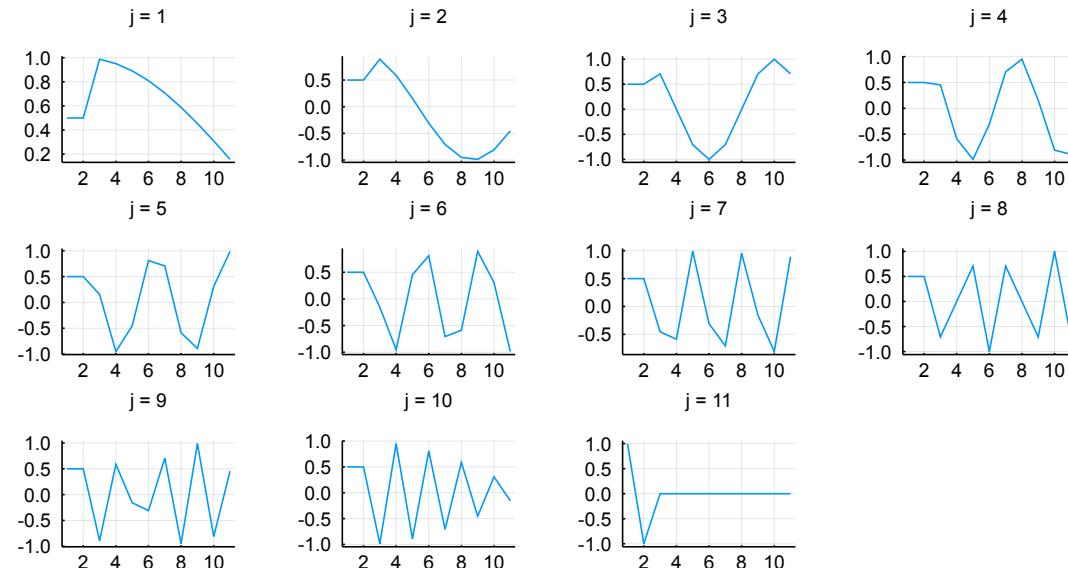
2.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-0.0	-0.0	0.0	0.0
0.0	1.8	-0.0	-0.0	0.0	-0.0	-0.0	-0.0	0.0	-0.0	-0.0
0.0	0.0	1.4	0.0	-0.0	0.0	0.0	0.0	-0.0	-0.0	-0.0
-0.0	0.0	0.0	0.9	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0
-0.0	0.0	-0.0	0.0	0.3	0.0	-0.0	-0.0	-0.0	0.0	-0.0
0.0	-0.0	0.0	-0.0	0.0	-0.3	0.0	0.0	0.0	-0.0	-0.0
0.0	0.0	0.0	0.0	-0.0	0.0	-0.9	0.0	0.0	0.0	-0.0
0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-1.4	0.0	-0.0	0.0
-0.0	0.0	0.0	0.0	0.0	0.0	-0.0	-0.0	-1.8	-0.0	0.0
0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	-2.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1x11 Array{Float64,2}:

1.98	1.78	1.41	0.91	0.31	-0.31	-0.91	-1.41	-1.78	-1.98	0.0
------	------	------	------	------	-------	-------	-------	-------	-------	-----

1x11 Array{Float64,2}:

1.98	1.78	1.41	0.91	0.31	-0.31	-0.91	-1.41	-1.78	-1.98	0.0
------	------	------	------	------	-------	-------	-------	-------	-------	-----



In [13]:

```

1 n = 12
2 A = adjacent_matrix_of_type_D(n)
3 V = eigenvectors_of_type_D(n)
4 α = eigenvalues_of_type_D(n)
5 display(A)
6 display(round.(V\.(A*V), digits=1))
7 display(round.(diag(V\.(A*V))', digits=2))
8 display(round.(α', digits=2))
9
10 PP = []
11 ▼ for j in 1:n
12     P = plot(V[:,j], title="j = $j", titlefontsize=8, legend=false)
13     push!(PP, P)
14 end
15 plot(PP..., size=(700, 360), legend=false) |> display

```

12×12 Symmetric{Int64,Array{Int64,2}}:

0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	1	0

12×12 Array{Float64,2}:

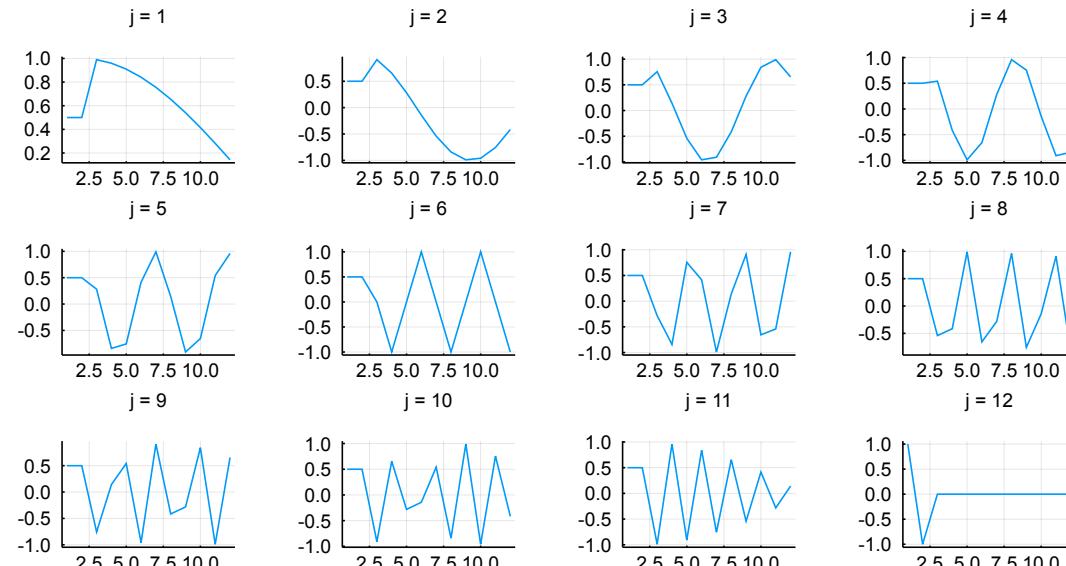
2.0	-0.0	0.0	0.0	0.0	-0.0	0.0	0.0	-0.0	0.0	0.0	0.0
0.0	1.8	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-0.0	-0.0	-0.0
0.0	0.0	1.5	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	-0.0	-0.0
-0.0	-0.0	0.0	1.1	0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0
-0.0	0.0	-0.0	0.0	0.6	0.0	0.0	0.0	-0.0	0.0	-0.0	-0.0
-0.0	-0.0	-0.0	-0.0	0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0
-0.0	0.0	0.0	0.0	-0.0	0.0	-0.0	-0.0	0.0	-0.0	0.0	-0.0
-0.0	-0.0	0.0	0.0	-0.0	0.0	-0.6	0.0	-0.0	0.0	-0.0	0.0
-0.0	-0.0	0.0	-0.0	0.0	-0.0	-0.0	-1.1	0.0	-0.0	0.0	-0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.5	0.0	0.0	0.0
-0.0	-0.0	0.0	-0.0	0.0	-0.0	-0.0	-0.0	-0.0	-1.8	-0.0	-0.0
0.0	0.0	-0.0	0.0	-0.0	0.0	0.0	0.0	0.0	-0.0	-2.0	0.0
-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0

1×12 Array{Float64,2}:

1.98	1.82	1.51	1.08	0.56	0.0	-0.56	-1.08	-1.51	-1.82	-1.98	0.0
------	------	------	------	------	-----	-------	-------	-------	-------	-------	-----

1×12 Array{Float64,2}:

1.98	1.82	1.51	1.08	0.56	0.0	-0.56	-1.08	-1.51	-1.82	-1.98	0.0
------	------	------	------	------	-----	-------	-------	-------	-------	-------	-----



2 古典アフィン型の場合

2.1 $A_{n-1}^{(1)}$ 型

$n \geq 3$ と仮定する. $A_{n-1}^{(1)}$ 型の隣接行列とは次の形の $n \times n$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 1 & & & 1 \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ 1 & & & 1 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0, 1, \dots, n-1\} \times \{0, 1, \dots, n-1\}$ 行列とみなす.

ベクトル $[x_k]_{k \in \mathbb{Z}} \neq 0$ が A_∞ 型隣接行列の固有値 α の固有ベクトルのとき, $[x_k]_{k=0}^{n-1}$ が $A_n^{(1)}$ 型の隣接行列の固有ベクトルになるための必要十分条件は周期境界条件 $x_{k+n} = x_k$ が成立していることである. このことと, A_∞ 型隣接行列の固有値・固有ベクトルに関する結果より以下が得られる. ζ を

$$\zeta = e^{i\theta} = e^{\frac{2\pi i}{n}}$$

と定める. このとき, $A_{n-1}^{(1)}$ 型隣接行列は以下の固有値 α_j と対応する固有ベクトル v_j を持つ:

$$\alpha_j = \zeta^j + \zeta^{-j} = 2 \cos \frac{2\pi j}{n}, \quad v_j = [\zeta^{jk}]_{k=0}^{n-1}.$$

ここで $j = 0, 1, \dots, n-1$ である.

$\alpha_{n-j} = \alpha_j$ となっていることに注意せよ. n が奇数のとき, $n = 2m+1$ とおくと,

$$\alpha_1 = \alpha_{2m}, \alpha_2 = \alpha_{2m-1}, \dots, \alpha_m = \alpha_{m+1}$$

であり, n が偶数のとき, $n = 2m$ とおくと,

$$\alpha_1 = \alpha_{2m-1}, \alpha_2 = \alpha_{2m-2}, \dots, \alpha_{m-1} = \alpha_{m+1}.$$

2.2 $C_n^{(1)}$ 型

$C_n^{(1)}$ 型の隣接行列とは次の形の $(n+1) \times (n+1)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 2 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & 0 & 1 \\ & & & & 2 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0, 1, \dots, n\} \times \{0, 1, \dots, n\}$ 行列とみなす.

ベクトル $v = [x_k]_{k=0}^{2n-1}$ が $A_{2n-1}^{(1)}$ 型隣接行列の固有値 α の固有ベクトルであるとする. そのとき, さらに $x_{2n-1} = x_1$ かつ $x_{n+1} = x_{n-1}$ ならば, ベクトル $[x_k]_{k=0}^n$ は $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルになる.

ゆえに, $C_n^{(1)}$ 型隣接行列は以下の固有値 α_j と対応する固有ベクトル v_j を持つ:

$$\alpha_j = 2 \cos \frac{j\pi}{n}, \quad v_j = \left[\cos \frac{kj\pi}{n} \right]_{k=0}^n.$$

2.3 $A_{2n+2}^{(2)}$ 型

$A_{2n+2}^{(2)}$ 型の隣接行列とは次の形の $(n+1) \times (n+1)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 1 & & & & \\ 2 & 0 & 1 & & & \\ & 1 & 0 & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & 0 & 1 \\ & & & & 2 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0, 1, \dots, n\} \times \{0, 1, \dots, n\}$ 行列とみなす.

ベクトル $v = [x_k]_{k=0}^n$ が $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルならば, v の最初の成分 x_0 を半分にしたものは $A_{2n+2}^{(2)}$ 型隣接行列の固有値 α の固有ベクトルになる.

2.4 $D_{n+2}^{(2)}$ 型

$D_{n+2}^{(2)}$ 型の隣接行列とは次の形の $(n+1) \times (n+1)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 1 & & & & \\ 2 & 0 & 1 & & & \\ & 1 & 0 & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & 0 & 2 \\ & & & & 1 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0, 1, \dots, n\} \times \{0, 1, \dots, n\}$ 行列とみなす.

ベクトル $v = [x_k]_{k=0}^n$ が $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルならば, v の最初の成分 x_0 と最後の成分 x_n を半分にしたものは $D_{n+2}^{(2)}$ 型隣接行列の固有値 α の固有ベクトルになる.

```
In [14]: 1 n = 5
          2 A = Matrix(Tridiagonal([ones(Int,n-1);2], zeros(Int, n+1), [2;ones(Int,n-1)]))
          3 display(A)
          4 eigA = eigen(A)
          5 display(round.(eigA.values, digits=3))
          6 V = eigA.vectors
          7 V = V/Diagonal(V[1,:])
          8 display(round.(V, digits=2))
```

```
6×6 Array{Int64,2}:
 0 2 0 0 0 0
 1 0 1 0 0 0
 0 1 0 1 0 0
 0 0 1 0 1 0
 0 0 0 1 0 1
 0 0 0 0 2 0

6-element Array{Float64,1}:
 -2.0
 -1.618
 -0.618
 0.618
 2.0
 1.618

6×6 Array{Float64,2}:
 1.0 1.0 1.0 1.0 1.0 1.0
 -1.0 -0.81 -0.31 0.31 1.0 0.81
 1.0 0.31 -0.81 -0.81 1.0 0.31
 -1.0 0.31 0.81 -0.81 1.0 -0.31
 1.0 -0.81 0.31 0.31 1.0 -0.81
 -1.0 1.0 -1.0 1.0 1.0 -1.0
```

In [15]:

```

1 n = 5
2 A = Matrix(Tridiagonal([2; ones(Int,n-2); 2], zeros(Int, n+1), ones(Int,n)))
3 display(A)
4 eigA = eigen(A)
5 display(round.(eigA.values, digits=3))
6 V = eigA.vectors
7 V = V/Diagonal(V[1,:])/2
8 display(round.(V, digits=2))

```

6×6 Array{Int64,2}:

0	1	0	0	0	0
2	0	1	0	0	0
0	1	0	1	0	0
0	0	1	0	1	0
0	0	0	1	0	1
0	0	0	0	2	0

6-element Array{Float64,1}:

-2.0
-1.618
-0.618
0.618
2.0
1.618

6×6 Array{Float64,2}:

0.5	0.5	0.5	0.5	0.5	0.5
-1.0	-0.81	-0.31	0.31	1.0	0.81
1.0	0.31	-0.81	-0.81	1.0	0.31
-1.0	0.31	0.81	-0.81	1.0	-0.31
1.0	-0.81	0.31	0.31	1.0	-0.81
-1.0	1.0	-1.0	1.0	1.0	-1.0

In [16]:

```

1 n = 5
2 A = Matrix(Tridiagonal([2;ones(Int,n-1)], zeros(Int, n+1), [ones(Int,n-1);2]))
3 display(A)
4 eigA = eigen(A)
5 display(round.(eigA.values, digits=3))
6 V = eigA.vectors
7 V = V/Diagonal(V[1,:])/2
8 display(round.(V, digits=2))

```

6×6 Array{Int64,2}:

0	1	0	0	0	0
2	0	1	0	0	0
0	1	0	1	0	0
0	0	1	0	1	0
0	0	0	1	0	2
0	0	0	0	1	0

6-element Array{Float64,1}:

-2.0
-1.618
-0.618
0.618
2.0
1.618

6×6 Array{Float64,2}:

0.5	0.5	0.5	0.5	0.5	0.5
-1.0	-0.81	-0.31	0.31	1.0	0.81
1.0	0.31	-0.81	-0.81	1.0	0.31
-1.0	0.31	0.81	-0.81	1.0	-0.31
1.0	-0.81	0.31	0.31	1.0	-0.81
-0.5	0.5	-0.5	0.5	0.5	-0.5

2.5 $A_{2n+3}^{(2)}$ 型

$A_{2n+3}^{(2)}$ 型の隣接行列とは次の形の $(n + 2) \times (n + 2)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ & & 1 & 0 & \ddots \\ & & & \ddots & \ddots & 1 \\ & & & & 1 & 0 & 1 \\ & & & & & 2 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0', 0, 1, \dots, n\} \times \{0', 0, 1, \dots, n\}$ 行列とみなす。

ベクトル $v = [x_k]_{k=0}^n$ が $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルならば, v の最初の成分 x_0 を半分にしたものと重複させて得られるベクトル

$$\begin{bmatrix} x_0/2 \\ x_0/2 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

は $A_{2n+3}^{(2)}$ 型の隣接行列の固有値 α の固有ベクトルになる。その他に

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

は $A_{2n+3}^{(2)}$ 型の隣接行列の固有値 0 の固有ベクトルになる。

2.6 $B_{n+1}^{(1)}$ 型

$B_{n+1}^{(1)}$ 型の隣接行列とは次の形の $(n+2) \times (n+2)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ & & 1 & 0 & \ddots \\ & & & \ddots & \ddots & 1 \\ & & & & 1 & 0 & 2 \\ & & & & & 1 & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0', 0, 1, \dots, n\} \times \{0', 0, 1, \dots, n\}$ 行列とみなす。

ベクトル $v = [x_k]_{k=0}^n$ が $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルならば, v の最初の成分 x_0 を半分にしたものと重複させ, 最後の成分 x_n を半分にして得られるベクトル

$$\begin{bmatrix} x_0/2 \\ x_0/2 \\ x_1 \\ \vdots \\ x_{n-1} \\ x_n/2 \end{bmatrix}$$

は $B_{n+1}^{(1)}$ 型の隣接行列の固有値 α の固有ベクトルになる。その他に

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

は $B_{n+1}^{(1)}$ 型の隣接行列の固有値 0 の固有ベクトルになる。

2.7 $D_{n+2}^{(1)}$ 型

$D_{n+2}^{(1)}$ 型の隣接行列とは次の形の $(n+3) \times (n+3)$ 行列のことであると定める:

$$\begin{bmatrix} 0 & 0 & 1 & & & & \\ 0 & 0 & 1 & & & & \\ 1 & 1 & 0 & 1 & & & \\ & 1 & 0 & & \ddots & & \\ & & \ddots & \ddots & 1 & & \\ & & & 1 & 0 & 1 & 1 \\ & & & & 1 & 0 & 0 \\ & & & & & 1 & 0 \\ & & & & & & 0 \end{bmatrix}.$$

以下ではこの行列を $\{0', 0, 1, \dots, n, n'\} \times \{0', 0, 1, \dots, n, n'\}$ 行列とみなす。

ベクトル $v = [x_k]_{k=0}^n$ が $C_n^{(1)}$ 型隣接行列の固有値 α の固有ベクトルならば, v の最初の成分 x_0 を半分にしたものと重複させ, 最後の成分 x_n を半分にして重複して得られるベクトル

$$\begin{bmatrix} x_0/2 \\ x_0/2 \\ x_1 \\ \vdots \\ x_{n-1} \\ x_n/2 \\ x_n/2 \end{bmatrix}$$

は $D_{n+2}^{(1)}$ 型の隣接行列の固有値 α の固有ベクトルになる。その他に

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ -1 \end{bmatrix},$$

は $B_{n+1}^{(1)}$ 型の隣接行列の固有値 0 の固有ベクトルになる。

3 Chebyshev多項式

3.1 Chebyshev多項式の定義

Chebyshev多項式 (チェビシェフ多項式) の話は本質的に三角函数の n 倍角の公式の話に過ぎない。そこでまず三角函数の n 倍角の公式を母函数の方法を使って計算してみよう。

$\theta, t \in \mathbb{R}, |t| < 1$ であるとする。このとき,

$$\sum_{n=0}^{\infty} e^{in\theta} t^n = \frac{1}{1 - e^{i\theta} t} = \frac{1 - e^{-i\theta} t}{(1 - e^{i\theta} t)(1 - e^{-i\theta} t)} = \frac{(1 - t \cos \theta) + it \sin \theta}{1 - 2t \cos \theta + t^2}.$$

ゆえに、両辺の実部と虚部を比較することによって次を得る:

$$\sum_{n=0}^{\infty} t^n \cos(n\theta) = \frac{1 - t \cos \theta}{1 - 2t \cos \theta + t^2},$$

$$\sum_{n=1}^{\infty} t^n \sin(n\theta) = \frac{t \sin \theta}{1 - 2t \cos \theta + t^2}.$$

後者の式の両辺を $t \sin \theta$ で割れば次が得られる:

$$\sum_{n=0}^{\infty} t^n \frac{\sin((n+1)\theta)}{\sin \theta} = \frac{1}{1 - 2t \cos \theta + t^2}.$$

ゆえに, t についてべき級数展開することによって, x の多項式 $T_n(x)$, $U_n(x)$ を

$$\frac{1 - xt}{1 - 2xt + t^2} = \sum_{n=0}^{\infty} T_n(x)t^n, \quad \frac{1}{1 - 2xt + t^2} = \sum_{n=0}^{\infty} U_n(x)t^n$$

と定義すると,

$$\cos(n\theta) = T_n(\cos \theta), \quad \frac{\sin((n+1)\theta)}{\sin \theta} = U_n(\cos \theta)$$

が得られる. $T_n(x)$ を第1種Chebyshev多項式と呼び, $U_n(x)$ を第2種Chebyshev多項式と呼ぶ.

3.2 Chebyshev多項式の具体形

$$\begin{aligned} \frac{1}{1 - 2xt + t^2} &= \sum_{m=0}^{\infty} (2xt - t^2)^m \\ &= \sum_{m=0}^{\infty} \sum_{j=0}^m (-1)^j \binom{m}{j} 2^{m-j} x^{m-j} t^{m+j} \\ &= \sum_{n=0}^{\infty} \left(\sum_{0 \leq j \leq n/2} (-1)^j \binom{n-j}{j} 2^{n-2j} x^{n-2j} \right) t^n \\ \frac{1 - xt}{1 - 2xt + t^2} &= (1 - xt) \frac{1}{1 - 2xt + t^2} \\ &= \sum_{n=0}^{\infty} \left(\sum_{0 \leq j \leq n/2} (-1)^j \binom{n-j}{j} 2^{n-2j} x^{n-2j} \right) t^n \\ &\quad + \sum_{n=0}^{\infty} \left(\sum_{0 \leq j \leq n/2} (-1)^{j+1} \binom{n-j}{j} 2^{n-2j} x^{n-2j+1} \right) t^{n+1} \\ &= 1 + \sum_{n=1}^{\infty} \left(\sum_{0 \leq j \leq n/2} (-1)^j \binom{n-j}{j} 2^{n-2j} x^{n-2j} \right) t^n \\ &\quad + \sum_{n=1}^{\infty} \left(\sum_{0 \leq j \leq (n-1)/2} (-1)^{j+1} \binom{n-j-1}{j} 2^{n-2j-1} x^{n-2j} \right) t^n \\ &= 1 + \sum_{n=1}^{\infty} \left(\frac{n}{2} \sum_{0 \leq j \leq n/2} \frac{(-1)^j}{n-j} \binom{n-j}{j} 2^{n-2j} x^{n-2j} \right) t^n \end{aligned}$$

より, $T_0(x) = U_0(x) = 1$ でかつ $n > 0$ のとき,

$$\begin{aligned} T_n(x) &= \frac{n}{2} \sum_{0 \leq j \leq n/2} \frac{(-1)^j}{n-j} \binom{n-j}{j} 2^{n-2j} x^{n-2j}, \\ U_n(x) &= \sum_{0 \leq j \leq n/2} (-1)^j \binom{n-j}{j} 2^{n-2j} x^{n-2j}. \end{aligned}$$

$n > 0$ のとき, $T_n(x)$, $U_n(x)$ はそれぞれ最高次の係数が 2^{n-1} , 2^n の多項式であり, それらの函数としての偶奇と n の偶奇は一致する.

In [17]:

```

1 ▾ function MyChebyshevT(n, x)
2     T = typeof(x)
3     (n/T(2)) * sum((-1)^j/T(n-j)*binomial(n-j,j)*(2x)^(n-2j) for j in 0:n÷2)
4 end
5
6 @vars x
7 N = 10
8 [MyChebyshevT(n, x) for n in 1:N] |> display
9 [ChebyshevT(n, x) for n in 1:N] |> display

```

$$\begin{bmatrix} x \\ 2x^2 - 1 \\ 4x^3 - 3x \\ 8x^4 - 8x^2 + 1 \\ 16x^5 - 20x^3 + 5x \\ 32x^6 - 48x^4 + 18x^2 - 1 \\ 64x^7 - 112x^5 + 56x^3 - 7x \\ 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\ 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ 2x^2 - 1 \\ 4x^3 - 3x \\ 8x^4 - 8x^2 + 1 \\ 16x^5 - 20x^3 + 5x \\ 32x^6 - 48x^4 + 18x^2 - 1 \\ 64x^7 - 112x^5 + 56x^3 - 7x \\ 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\ 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \end{bmatrix}$$

In [18]:

```

1 ▼ function MyChebyshevU(n, x)
2     sum((-1)^j*binomial(n-j,j)*(2x)^(n-2j)) for j in 0:n÷2
3 end
4
5 @vars x
6 N = 10
7 [MyChebyshevU(n, x) for n in 1:N] |> display
8 [ChebyshevU(n, x) for n in 1:N] |> display

```

$$\begin{bmatrix} 2x \\ 4x^2 - 1 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 + 1 \\ 32x^5 - 32x^3 + 6x \\ 64x^6 - 80x^4 + 24x^2 - 1 \\ 128x^7 - 192x^5 + 80x^3 - 8x \\ 256x^8 - 448x^6 + 240x^4 - 40x^2 + 1 \\ 512x^9 - 1024x^7 + 672x^5 - 160x^3 + 10x \\ 1024x^{10} - 2304x^8 + 1792x^6 - 560x^4 + 60x^2 - 1 \end{bmatrix}$$

$$\begin{bmatrix} 2x \\ 4x^2 - 1 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 + 1 \\ 32x^5 - 32x^3 + 6x \\ 64x^6 - 80x^4 + 24x^2 - 1 \\ 128x^7 - 192x^5 + 80x^3 - 8x \\ 256x^8 - 448x^6 + 240x^4 - 40x^2 + 1 \\ 512x^9 - 1024x^7 + 672x^5 - 160x^3 + 10x \\ 1024x^{10} - 2304x^8 + 1792x^6 - 560x^4 + 60x^2 - 1 \end{bmatrix}$$

3.3 Chebyshev多項式の因数分解と隣接行列の特性多項式の関係

3.3.1 第1種Chebyshev多項式の場合

θ_j 達を

$$\theta_j = \frac{(2j+1)\pi}{2n} \quad (j = 0, 1, \dots, n-1)$$

と定めると, $\cos(n\theta_j) = 0$ となり, $\cos \theta_j$ は互いに異なる. ゆえに $\cos(n\theta) = T_n(\cos \theta)$ かつ $T_n(x)$ が最高次の係数が 2^{n-1} の n 次多項式であることより,

$$T_n(x) = 2^{n-1} \prod_{j=0}^{n-1} (x - \cos \theta_j) = 2^{n-1} \prod_{j=0}^{n-1} (x + \cos \theta_j)$$

を得る. この公式の後者の等号は $\cos \theta_{n-1-j} = \cos(\pi - \theta_j) = -\cos \theta_j$ から得られる.

一方, C_n 型の $n \times n$ の隣接行列

$$\begin{bmatrix} 0 & 2 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix}$$

の固有値の全体は $2 \cos \theta_j$ ($j = 0, 1, \dots, n-1$) だったので,

$$\begin{vmatrix} 2x & 2 & & \\ 1 & 2x & 1 & \\ & 1 & 2x & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 2x \end{vmatrix} = \begin{vmatrix} 2x & -2 & & \\ -1 & 2x & -1 & \\ & -1 & 2x & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2x \end{vmatrix} = 2T_n(x)$$

左辺は C_n 型隣接行列の特性多項式 A の -1 倍の特性多項式に $2x$ を代入して得られる $|2xE + A|$ であり, A と $-A$ の固有値の全体は一致するので, それは A の特性多項式に $2x$ を代入したもの $|2xE - A|$ に等しいことがわかり, 零点と最高次の係数の一致によって2つ目の等号も成立する. この公式の両辺を 2で割ると,

$$\begin{vmatrix} x & 1 & & \\ 1 & 2x & 1 & \\ & 1 & 2x & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 2x \end{vmatrix} = T_n(x)$$

も得られる. B_n 型隣接行列は C_n 型隣接行列の転置に等しく, 行列式は転置で不变なので, 以上と同じ結果が B_n 型隣接行列についても得られる. D_{n+1} 型の $(n+1) \times (n+1)$ の隣接行列について同様にすると以下の公式も得られる:

$$\begin{vmatrix} 2x & 0 & 1 & & \\ 0 & 2x & 1 & & \\ 1 & 1 & 2x & 1 & \\ & 1 & 2x & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 2x \end{vmatrix} = 4xT_n(x).$$

In [19]:

```

1 ▼  function charpoly_C(n,x)
2      @assert n ≥ 1
3 ▼      if n ≥ 2
4          Diagonal(fill(2x, n)) + adjacent_matrix_of_type_C(n)
5 ▼      else
6          hcat([2x])
7      end
8  end
9
10 @vars x
11 charpoly_C(5,x) |> display
12
13 N = 10
14 [det(charpoly_C(n,x)).expand() for n in 1:N] |> display
15 [2ChebyshevT(n,x) for n in 1:N] |> display

```

$$\begin{bmatrix} 2x & 2 & 0 & 0 & 0 \\ 1 & 2x & 1 & 0 & 0 \\ 0 & 1 & 2x & 1 & 0 \\ 0 & 0 & 1 & 2x & 1 \\ 0 & 0 & 0 & 1 & 2x \end{bmatrix}$$

$$\begin{bmatrix} 2x \\ 4x^2 - 2 \\ 8x^3 - 6x \\ 16x^4 - 16x^2 + 2 \\ 32x^5 - 40x^3 + 10x \\ 64x^6 - 96x^4 + 36x^2 - 2 \\ 128x^7 - 224x^5 + 112x^3 - 14x \\ 256x^8 - 512x^6 + 320x^4 - 64x^2 + 2 \\ 512x^9 - 1152x^7 + 864x^5 - 240x^3 + 18x \\ 1024x^{10} - 2560x^8 + 2240x^6 - 800x^4 + 100x^2 - 2 \end{bmatrix}$$

$$\begin{bmatrix} 2x \\ 4x^2 - 2 \\ 8x^3 - 6x \\ 16x^4 - 16x^2 + 2 \\ 32x^5 - 40x^3 + 10x \\ 64x^6 - 96x^4 + 36x^2 - 2 \\ 128x^7 - 224x^5 + 112x^3 - 14x \\ 256x^8 - 512x^6 + 320x^4 - 64x^2 + 2 \\ 512x^9 - 1152x^7 + 864x^5 - 240x^3 + 18x \\ 1024x^{10} - 2560x^8 + 2240x^6 - 800x^4 + 100x^2 - 2 \end{bmatrix}$$

In [20]:

```

1 ▼ function charpoly_halfC(n,x)
2     @assert n ≥ 1
3 ▼     if n ≥ 2
4         Diagonal([x;fill(2x, n-1)]) + adjacent_matrix_of_type_A(n)
5 ▼     else
6         hcat([x])
7     end
8 end
9
10 @vars x
11 charpoly_halfC(5,x) ▷ display
12
13 N = 10
14 [det(charpoly_halfC(n,x)).expand() for n in 1:N] ▷ display
15 [ChebyshevT(n,x) for n in 1:N] ▷ display

```

$$\begin{bmatrix} x & 1 & 0 & 0 & 0 \\ 1 & 2x & 1 & 0 & 0 \\ 0 & 1 & 2x & 1 & 0 \\ 0 & 0 & 1 & 2x & 1 \\ 0 & 0 & 0 & 1 & 2x \end{bmatrix}$$

$$\begin{bmatrix} x \\ 2x^2 - 1 \\ 4x^3 - 3x \\ 8x^4 - 8x^2 + 1 \\ 16x^5 - 20x^3 + 5x \\ 32x^6 - 48x^4 + 18x^2 - 1 \\ 64x^7 - 112x^5 + 56x^3 - 7x \\ 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\ 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ 2x^2 - 1 \\ 4x^3 - 3x \\ 8x^4 - 8x^2 + 1 \\ 16x^5 - 20x^3 + 5x \\ 32x^6 - 48x^4 + 18x^2 - 1 \\ 64x^7 - 112x^5 + 56x^3 - 7x \\ 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\ 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \end{bmatrix}$$

In [21]:

```

1 ▼ function charpoly_D(n,x)
2     @assert n ≥ 2
3 ▼     if n ≥ 3
4         Diagonal(fill(2x, n)) + adjacent_matrix_of_type_D(n)
5 ▼     else
6         Diagonal(fill(2x, 2))
7     end
8 end
9
10 @vars x
11 charpoly_D(5,x) ▶ display
12
13 N = 10
14 [det(charpoly_D(n+1,x)).simplify().expand() for n in 1:N] ▶ display
15 [(4x*ChebyshevT(n,x)).expand() for n in 1:N] ▶ display

```

$$\begin{bmatrix} 2x & 0 & 1 & 0 & 0 \\ 0 & 2x & 1 & 0 & 0 \\ 1 & 1 & 2x & 1 & 0 \\ 0 & 0 & 1 & 2x & 1 \\ 0 & 0 & 0 & 1 & 2x \end{bmatrix}$$

$$\begin{bmatrix} 4x^2 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 \\ 32x^5 - 32x^3 + 4x \\ 64x^6 - 80x^4 + 20x^2 \\ 128x^7 - 192x^5 + 72x^3 - 4x \\ 256x^8 - 448x^6 + 224x^4 - 28x^2 \\ 512x^9 - 1024x^7 + 640x^5 - 128x^3 + 4x \\ 1024x^{10} - 2304x^8 + 1728x^6 - 480x^4 + 36x^2 \\ 2048x^{11} - 5120x^9 + 4480x^7 - 1600x^5 + 200x^3 - 4x \end{bmatrix}$$

$$\begin{bmatrix} 4x^2 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 \\ 32x^5 - 32x^3 + 4x \\ 64x^6 - 80x^4 + 20x^2 \\ 128x^7 - 192x^5 + 72x^3 - 4x \\ 256x^8 - 448x^6 + 224x^4 - 28x^2 \\ 512x^9 - 1024x^7 + 640x^5 - 128x^3 + 4x \\ 1024x^{10} - 2304x^8 + 1728x^6 - 480x^4 + 36x^2 \\ 2048x^{11} - 5120x^9 + 4480x^7 - 1600x^5 + 200x^3 - 4x \end{bmatrix}$$

3.3.2 第2種Chebyshev多項式の場合

θ_j 達を

$$\theta_j = \frac{j\pi}{n+1} \quad (j = 1, 2, \dots, n)$$

と定めると, $\sin((n+1)\theta_j) = 0$ となり, $\cos \theta_j$ は互いに異なる. ゆえに $\sin((n+1)\theta)/\sin \theta = U_n(\cos \theta)$ かつ $U_n(x)$ が最高次の係数が 2^n の n 次多項式であることより,

$$U_n(x) = 2^n \prod_{j=1}^n (x - \cos \theta_j) = 2^n \prod_{j=1}^n (x + \cos \theta_j)$$

を得る. この公式の後者の等号は $\cos \theta_{n+1-j} = \cos(\pi - \theta_j) = -\cos \theta_j$ から得られる.

一方, A_n 型隣接行列

$$\begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{bmatrix}$$

の固有値の全体は $2 \cos \theta_j$ ($j = 1, 2, \dots, n$) だったので、最高次の係数と零点の一致によって

$$\begin{bmatrix} 2x & 1 & & & \\ 1 & 2x & 1 & & \\ & 1 & 2x & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 2x \end{bmatrix} = \begin{bmatrix} 2x & -1 & & & \\ -1 & 2x & -1 & & \\ & -1 & 2x & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2x \end{bmatrix} = U_n(x)$$

が得られる。

```
In [22]: 1 ▼ function charpoly_A(n,x)
2 ▼   if n ≥ 2
3     Diagonal(fill(2x, n)) + adjacent_matrix_of_type_A(n)
4   else
5     hcat([2x])
6   end
7 end
8
9 @vars x
10 charpoly_A(5,x) |> display
11
12 N = 10
13 [det(charpoly_A(n,x)).simplify().expand() for n in 2:N] |> display
14 [ChebyshevU(n,x) for n in 2:N] |> display
```

$$\begin{bmatrix} 2x & 1 & 0 & 0 & 0 \\ 1 & 2x & 1 & 0 & 0 \\ 0 & 1 & 2x & 1 & 0 \\ 0 & 0 & 1 & 2x & 1 \\ 0 & 0 & 0 & 1 & 2x \end{bmatrix}$$

$$\begin{bmatrix} 4x^2 - 1 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 + 1 \\ 32x^5 - 32x^3 + 6x \\ 64x^6 - 80x^4 + 24x^2 - 1 \\ 128x^7 - 192x^5 + 80x^3 - 8x \\ 256x^8 - 448x^6 + 240x^4 - 40x^2 + 1 \\ 512x^9 - 1024x^7 + 672x^5 - 160x^3 + 10x \\ 1024x^{10} - 2304x^8 + 1792x^6 - 560x^4 + 60x^2 - 1 \end{bmatrix}$$

$$\begin{bmatrix} 4x^2 - 1 \\ 8x^3 - 4x \\ 16x^4 - 12x^2 + 1 \\ 32x^5 - 32x^3 + 6x \\ 64x^6 - 80x^4 + 24x^2 - 1 \\ 128x^7 - 192x^5 + 80x^3 - 8x \\ 256x^8 - 448x^6 + 240x^4 - 40x^2 + 1 \\ 512x^9 - 1024x^7 + 672x^5 - 160x^3 + 10x \\ 1024x^{10} - 2304x^8 + 1792x^6 - 560x^4 + 60x^2 - 1 \end{bmatrix}$$

3.4 三角函数の無限積表示

3.4.1 \cos の無限積表示

n は偶数であるとし, $n = 2m$ とおくと, 上の方で示した結果より,

$$\begin{aligned}\cos(2mx) &= 2^{2m-1} \prod_{j=0}^{m-1} \left(\cos^2 x - \cos^2 \frac{(2j+1)\pi}{4m} \right) \\ &= (-1)^m 2^{2m-1} \prod_{j=0}^{m-1} \left(\sin^2 x - \sin^2 \frac{(2j+1)\pi}{4m} \right).\end{aligned}$$

この等式の両辺を $x = 0$ とおいた場合の等式の両辺で割ると次が得られる:

$$\cos(2mx) = \prod_{j=0}^{m-1} \left(1 - \frac{\sin^2 x}{\sin^2 \frac{(2j+1)\pi}{4m}} \right).$$

これに $x = \frac{\pi s}{4m}$ を代入すると,

$$\cos \frac{\pi s}{2} = \prod_{j=0}^{m-1} \left(1 - \frac{\sin^2 \frac{\pi s}{4m}}{\sin^2 \frac{(2j+1)\pi}{4m}} \right).$$

この等式の $m \rightarrow \infty$ の極限で次が得られる:

$$\cos \frac{\pi s}{2} = \prod_{j=0}^{\infty} \left(1 - \frac{s^2}{(2j+1)^2} \right) = \left(1 - \frac{s^2}{1^2} \right) \left(1 - \frac{s^2}{3^2} \right) \left(1 - \frac{s^2}{5^2} \right) \dots$$

3.4.2 sin の無限積表示

n は偶数であるとし, $n = 2m$ とおくと, 上の方で示した結果より,

$$\begin{aligned}\frac{\sin((2m+1)x)}{\sin x} &= 2^{2m} \prod_{j=1}^m \left(\cos^2 x - \cos^2 \frac{j\pi}{2m+1} \right) \\ &= (-1)^m 2^{2m} \prod_{j=1}^m \left(\sin^2 x - \sin^2 \frac{j\pi}{2m+1} \right).\end{aligned}$$

この等式の $x \rightarrow 0$ での極限の両辺でこの等式の両辺をそれぞれ割ると次が得られる:

$$\frac{\sin((2m+1)x)}{(2m+1) \sin x} = \prod_{j=1}^m \left(1 - \frac{\sin^2 x}{\sin^2 \frac{j\pi}{2m+1}} \right).$$

これに $x = \frac{\pi s}{2m+1}$ を代入すると,

$$\frac{\sin(\pi s)}{(2m+1) \sin \frac{\pi s}{2m+1}} = \prod_{j=1}^m \left(1 - \frac{\sin^2 \frac{\pi s}{2m+1}}{\sin^2 \frac{j\pi}{2m+1}} \right).$$

この等式の $m \rightarrow \infty$ での極限で次が得られる:

$$\frac{\sin(\pi s)}{\pi s} = \prod_{j=1}^{\infty} \left(1 - \frac{s^2}{j^2} \right) = \left(1 - \frac{s^2}{1^2} \right) \left(1 - \frac{s^2}{2^2} \right) \left(1 - \frac{s^2}{3^2} \right) \dots$$