Řešení 2D Schrödingerovy rovnice Lanczosovou diagonalizací

Tento dokument je výpočetní notebook pro Pluto.jl. Jeho zdrojová podoba je veřejně dostupná zde. Omluvte, prosím, nedokonalé stránkování, které zatím Pluto neumí opravit.

Vizte také:

- repozitář se všemi variantami kódu: https://github.com/Singond/Nummet-II-task;
- mírně upravený notebook pro JuliaHub, který můžete pustit online bez nutnosti instalace Julie a balíku `Pluto.jl`: https://juliahub.com/ui/Notebooks/Singond/School/schrodinger_makie.nb.jl.

```
begin
    import Pkg
    Pkg.activate(Base.current_project())
    Pkg.instantiate()

    using LinearAlgebra
    using SparseArrays

    using Plots
end
```

lanczos

Return a tridiagonal matrix `T` and orthonormal matrix `V` such that `T = V' * A * V`.

```
"""
Return a tridiagonal matrix `T` and orthonormal matrix `V`
such that `T = V' * A * V`.
"""
function lanczos(A, m, v)
    if ndims(A) != 2 || size(A)[1] != size(A)[2]
        error("A must be a square matrix")
    end
    n = size(A, 1)
    if (m > n)
        error("m must be at most size of A")
    end
    α = zeros(m)
    β = zeros(m)
    V = zeros(n, m)

    v = v ./ norm(v)
    V[:,1] = v
    w = A * v
    α[1] = w' * v
    w = w - α[1] * v

    for j in 2:m
        β[j] = norm(w)
        if β[j] != 0
            V[:,j] = w ./ β[j]
        else
            error("β == 0, j = $j")
            v = randn(n)
            V[:,j] = v ./ norm(v)
        end
        w = A * V[:,j]
        α[j] = w' * V[:,j]
        w = w - α[j] * V[:,j] - β[j] * V[:,j-1]
    end
    T = SymTridiagonal(α, β[2:end])
    T, V
end
```

```
lanczos (generic function with 2 methods)
```

```
lanczos(A, m) = lanczos(A, m, randn(size(A, 1)))
```

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

```
lanczos(A) = lanczos(A, round(Int, size(A, 1) / 2))
```

`eigenvectors`

Return the first `m` eigenvectors of matrix `A`.

This implementation is based on the Wikipedia article on Lanczos method.

```
"""
Return the first `m` eigenvectors of matrix `A`.

This implementation is based on the Wikipedia article on Lanczos method.
"""
function eigenvectors(A, args...)
    T, V = lanczos(A, args...)
    t = LinearAlgebra.eigvecs(T)
    V * t
end
```

Planckova konstanta v Js:

```
const global planckr = 1.054571817E-34;
```

Hmotnost elektronu v kg:

```
const global electronmass = 9.1093837E-31;
```

Elementární náboj v C:

```
const global elemcharge = 1.602176634E-19;
```

Pomocné funkce:

```
diff2 (generic function with 1 method)
```

```
function diff2(indices)
    n = maximum(indices)::Int
    ni = size(indices, 1)
    nj = size(indices, 2)
    D = spzeros(n, n)
    for i in 1:ni, j in 1:nj
        c = indices[i,j]
        D[c,c] -= 4
        if i > 1
            D[indices[i-1,j],c] += 1
        end
        if i < ni
            D[indices[i+1,j],c] += 1
        end
        if j > 1
            D[indices[i,j-1],c] += 1
        end
        if j < nj
            D[indices[i,j+1],c] += 1
        end
    end
    D
end
```

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

```
function hamiltonian(x, y, potential::Function; mass = 1, Δ = 1)
    D = diff2(LinearIndices((length(x), length(y))))
    T = (-planckr^2 / 2mass) * D / Δ^2
    V = potential.(x', y)
    T, V
end
```

Simulace probíhá na čtvercové oblasti o rozměrech $L \times L$ rozdělené na $N \times N$ uzlů.

```
N = 201
```

```
1 N = 201
```

```
L = 10
```

```
1 L = 10   # [nm]
```

Pomocné vektory x a y představující prostorové souřadnice:

```
x =
201-element LinRange{Float64, Int64}:
 -5.0, -4.95, -4.9, -4.85, -4.8, -4.75, -4.7, …, 4.7, 4.75, 4.8, 4.85, 4.9, 4.95, 5.0
```

```
1 x = LinRange(-L/2, L/2, N)
```

```
y =
201-element LinRange{Float64, Int64}:
 -5.0, -4.95, -4.9, -4.85, -4.8, -4.75, -4.7, …, 4.7, 4.75, 4.8, 4.85, 4.9, 4.95, 5.0
```

```
1 y = LinRange(-L/2, L/2, N)
```

Diference x (též y):

```
Δ = 0.05000000000000071
```

```
1 Δ = x[2] - x[1]
```

Obecné nastavení grafů:

```
1 default(c=cgrad(:viridis, rev=true))
```

Gaussovský potenciál

První případ je rotačně symetrický gaussovský potenciál s grupou symetrie O(2) vyjádřený vztahem:

$$V(x, y) = -V_0 \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

kde $V_0 = 2\,\mathrm{eV}$ a $\sigma = 2\,\mathrm{nm}$.

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

```
1 function potential_gauss(x, y, V0, σ)
2     -V0 * exp(-(x^2 + y^2) / 2σ^2)
3 end
```

```
potential_gauss (generic function with 2 methods)
```

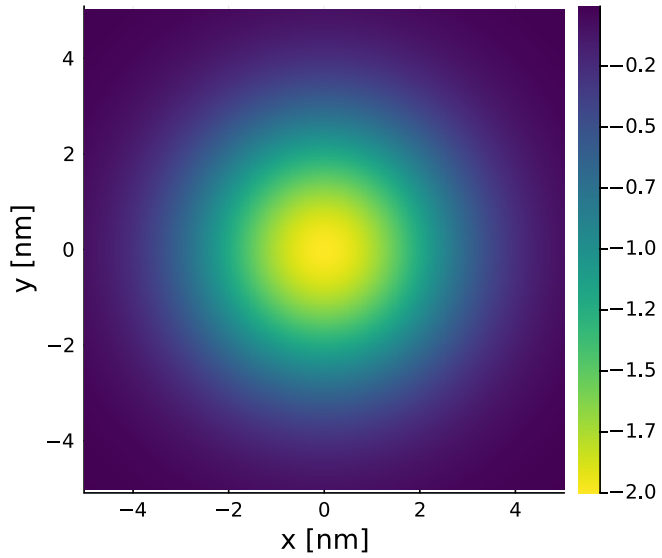
```
1 potential_gauss(x, y) = potential_gauss(x, y, 2, 2)
```

Složky hamiltoniánu:

```
(40401×40401 SparseMatrixCSC{Float64, Int64} with 201201 stored entries:, 201×201 Matrix{
```

```
1 Tg, Vg = hamiltonian(x, y, potential_gauss, Δ=Δ, mass=electronmass)
```

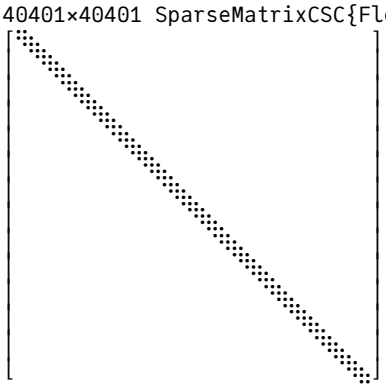
potenciál V_g




```
with(ratio=1, title="potenciál V_g", size=(400,430)) do
    heatmap(x, y, Vg, xlabel="x [nm]", ylabel="y [nm]")
end
```

Hamiltonián:

```
Hg = 40401×40401 SparseMatrixCSC{Float64, Int64} with 201201 stored entries:
```



```
Hg = Tg * (1E18 / elemcharge) + spdiagm(Vg[:])
```

Vlastní vektory hamiltoniánu spočtené zvolenou metodou:

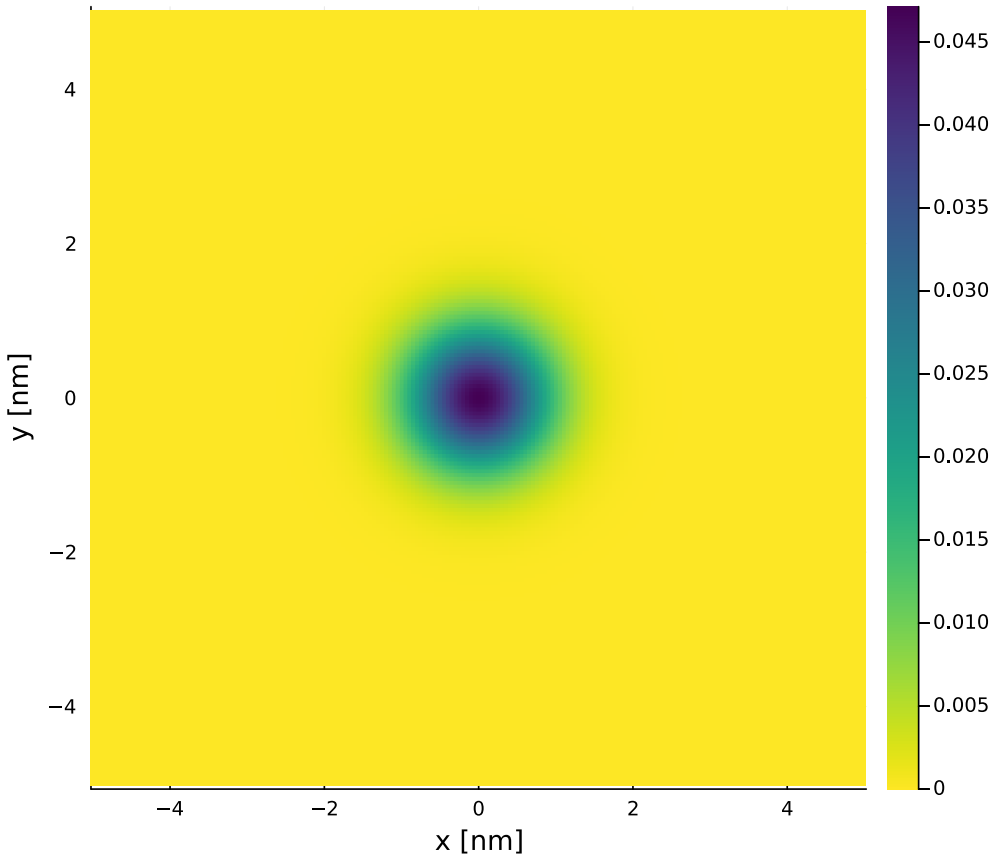
```
eg = 40401×800 Matrix{Float64}:
 -2.63187e-18  -9.15758e-18  -1.1363e-17   …  -0.000113313   8.35698e-5
  9.79418e-18  -1.15213e-17  -1.87375e-17      0.000226063  -0.000166724
 -3.49857e-18  -3.27337e-17  -1.37048e-17     -0.000337696   0.000249055
  1.59089e-19  -4.29004e-17  -2.34494e-17      0.000447669  -0.000330162
 -1.20227e-17  -5.41417e-17   1.04569e-17     -0.000555458   0.000409657
  2.46969e-18  -3.15274e-17   2.40966e-17  …   0.000660558  -0.00048717
 -1.78786e-18  -3.62857e-19   2.21819e-17     -0.000762489   0.000562346
  ⋮                                      ⋱
  2.38936e-17  -3.29397e-17   6.03041e-17  …  -0.000660558  -0.00048717
  3.0039e-17   -1.42642e-17   7.98985e-17      0.000555458   0.000409657
  3.25681e-17   1.01622e-17   8.22113e-17     -0.000447669  -0.000330162
  2.86097e-17   5.35178e-18   5.89387e-17      0.000337696   0.000249055
  3.19157e-17  -7.92107e-18   4.35123e-17     -0.000226063  -0.000166724
  1.02211e-17  -1.29801e-17   1.02012e-17  …   0.000113313   8.35698e-5
```

```
eg = eigenvectors(Hg, 800)
```

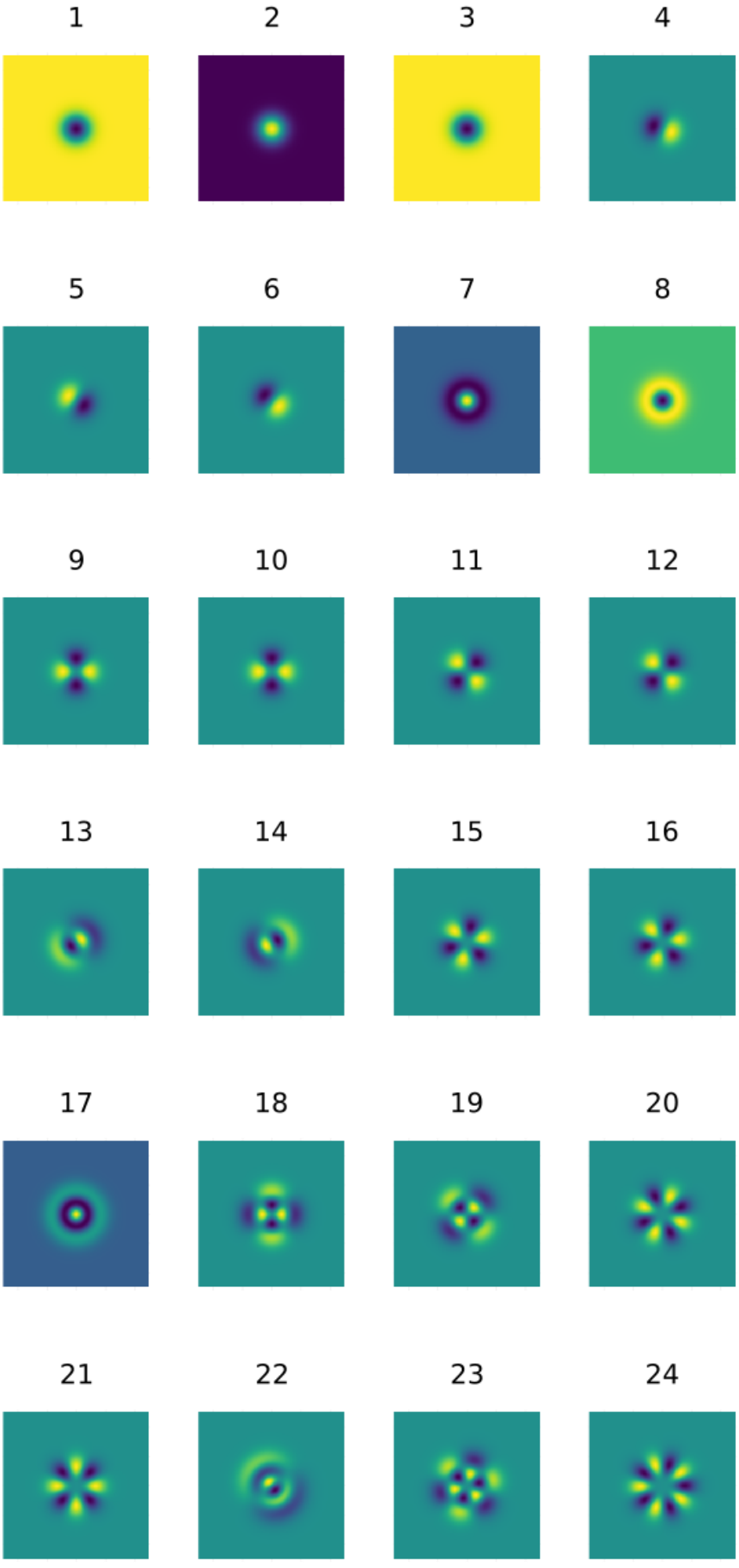
```
erg = 201×201×800 Array{Float64, 3}:
[:, :, 1] =
 -2.63187e-18   1.16579e-17   3.87601e-17  …  -3.39975e-20   1.62465e-17
  9.79418e-18   3.26087e-17   5.22191e-17     -1.65049e-17   6.47257e-18
 -3.49857e-18   3.51672e-17   4.59007e-17     -3.4556e-17   -5.95325e-18
  1.59089e-19   1.04165e-17   2.61615e-17     -5.18468e-17  -1.52666e-17
 -1.20227e-17   5.32419e-18   2.91852e-17     -6.17665e-17  -1.87263e-17
  2.46969e-18   1.68608e-17   3.06282e-17  …  -6.3827e-17   -2.42659e-17
 -1.78786e-18   3.14643e-17   4.38701e-17     -4.29093e-17  -1.6483e-17
  ⋮                                       ⋱                ⋮
 -1.88521e-17  -2.73545e-17  -3.61406e-17  …   4.85967e-17   2.38936e-17
 -8.74468e-18  -3.88132e-17  -1.46402e-17      4.14633e-17   3.0039e-17
 -7.15088e-18  -5.65505e-18  -9.98942e-18      5.17191e-17   3.25681e-17
  1.25297e-17   1.70159e-17   2.40336e-17      7.13252e-17   2.86097e-17
  1.12544e-17   3.45446e-17   4.62413e-17      5.4676e-17    3.19157e-17
  2.7464e-17    9.43271e-18   2.71498e-17  …   2.18504e-17   1.02211e-17

[:, :, 2] =
 -9.15758e-18   4.49349e-18   2.87685e-17  …  -2.2039e-17   -1.76062e-17
 -1.15213e-17  -1.00618e-17   2.03948e-17     -3.62691e-17  -2.09996e-17
 -3.27337e-17  -2.99921e-17  -1.18447e-17     -2.43777e-17  -2.39697e-17
 -4.29004e-17  -5.74657e-17  -6.05688e-17     -4.75715e-17  -1.10077e-17
 -5.41417e-17  -7.30472e-17  -9.71177e-17     -5.63343e-17  -1.37927e-17
 -3.15274e-17  -8.06282e-17  -1.0235e-16   …  -7.45784e-17  -4.08345e-17
 -3.62857e-19  -6.55265e-17  -7.35076e-17     -7.02339e-17  -6.52851e-17
  ⋮                                       ⋱                ⋮
 -4.6089e-17   -1.13943e-16  -1.4803e-16   …  -3.30675e-17  -3.29397e-17
 -5.21305e-17  -9.87899e-17  -1.45526e-16     -2.34408e-17  -1.42642e-17
 -5.59938e-17  -9.50636e-17  -1.10493e-16     -6.84528e-18   1.01622e-17
 -3.81996e-17  -3.50832e-17  -6.43002e-17     -5.09568e-18   5.35178e-18
 -1.77653e-17  -3.96574e-17  -1.89121e-17     -1.73355e-17  -7.92107e-18
 -5.00774e-18   3.9651e-18   -1.0015e-18   …  -2.41888e-17  -1.29801e-17
```

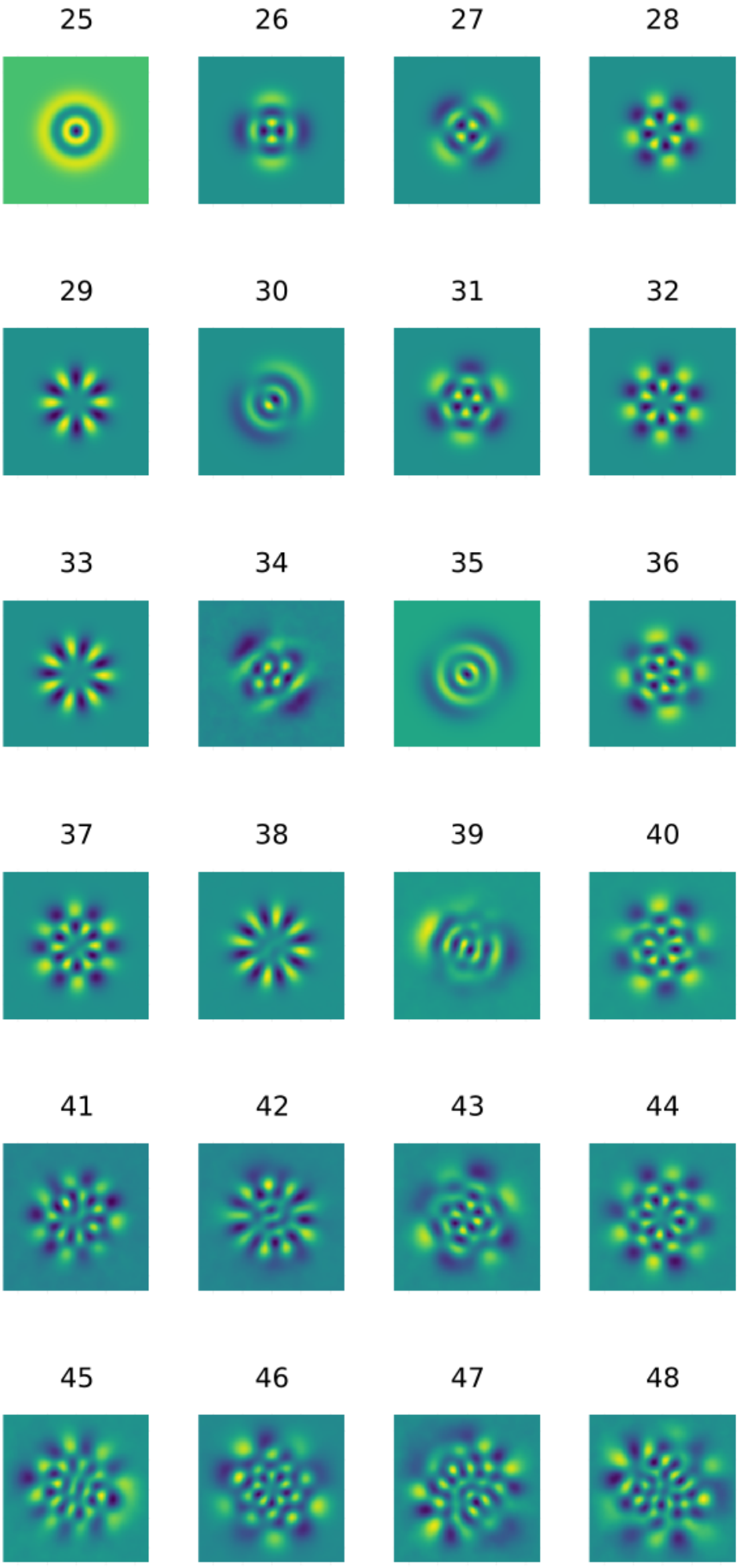
```
1 erg = reshape(eg, N, N, :)
```

k = 1




```
with(ratio=1, title="vlastní vektor $(kg)", size=(600,630)) do
    heatmap(x, y, erg[:,:,kg],
        xlabel="x [nm]", ylabel="y [nm]")
end
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```
with(ratio=1, size=(600,2500), plot_title="", legend=false, showaxis=false) do
    plotsb = map(1:48) do k
        heatmap(x, y, erg[:,:,k], title="$k")
    end
    plot(plotsb..., layout=(12,4))
end
```

Potenciál molekuly benzenu

Druhý případ je potenciál s grupou symetrie C_{6v}, který připomíná molekulu benzenu:

$$V(x,y) = -V_0 \sum_{n=1}^{6} \exp\left[-\frac{(x - R\cos\frac{n\pi}{3})^2 + (y - R\sin\frac{n\pi}{3})^2}{2\sigma^2}\right]$$

kde $V_0 = 2\,\mathrm{eV}$, $\sigma = 0{,}8\,\mathrm{nm}$ a $R = 2\,\mathrm{nm}$.

```
potential_c6v (generic function with 1 method)
```

```
function potential_c6v(x, y, V0, σ, R)
    v = 0
    for n in [1 2 3 4 5 6]
        s, c = sincospi(n / 3)
        v += exp(- ((x - R * c)^2 + (y - R * s)^2) / 2σ^2)
    end
    -V0 * v
end
```

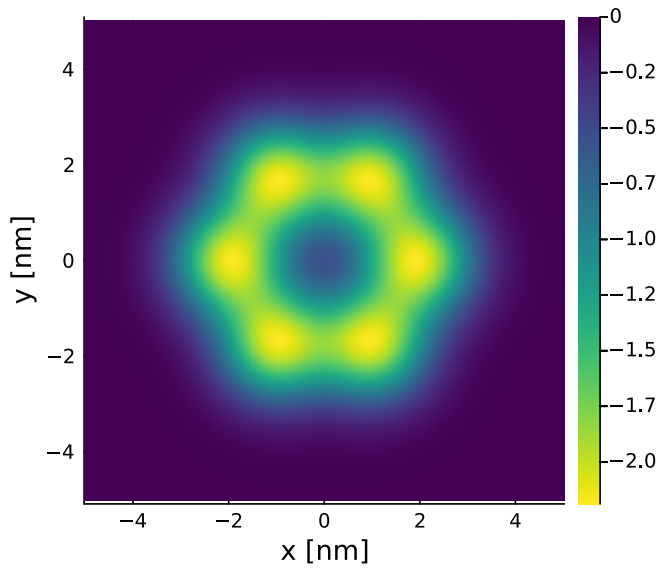
```
potential_c6v (generic function with 2 methods)
```

```
potential_c6v(x, y) = potential_c6v(x, y, 2, 0.8, 2)
```

Složky hamiltoniánu:

```
(40401×40401 SparseMatrixCSC{Float64, Int64} with 201201 stored entries:, 201×201 Matrix{
```

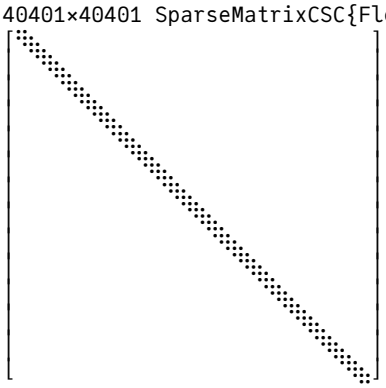
```
Tb, Vb = hamiltonian(x, y, potential_c6v, Δ=Δ, mass=electronmass)
```




```
with(ratio=1, title="potenciál V_b", size=(400,430)) do
    heatmap(x, y, Vb, xlabel="x [nm]", ylabel="y [nm]")
end
```

Hamiltonián:

```
Hb = 40401×40401 SparseMatrixCSC{Float64, Int64} with 201201 stored entries:
```



```
Hb = Tb * (1E18 / elemcharge) + spdiagm(Vb[:])
```

Vlastní vektory hamiltoniánu spočtené zvolenou metodou:

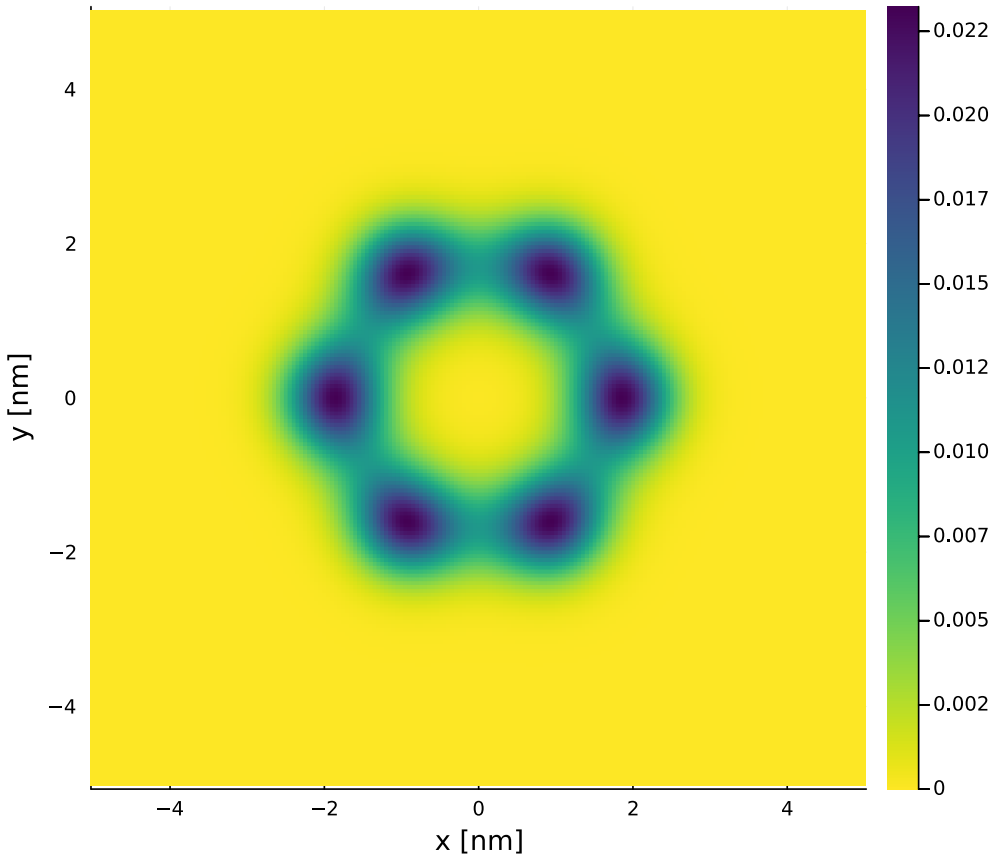
```
eb =
40401×800 Matrix{Float64}:
 -1.58931e-13  -3.26475e-10   3.09624e-17   2.04258e-7  …  -3.5043e-5    -3.46361e-5
 -3.36586e-13  -6.91442e-10   9.29952e-17   4.32521e-7      7.00617e-5     6.92088e-5
 -1.65084e-13  -3.39339e-10   1.35862e-16   2.12303e-7     -0.000104855   -0.000103613
 -1.28191e-13  -2.63668e-10   2.03664e-16   1.64841e-7      0.000139373    0.000137782
  3.38185e-14   6.88897e-11   2.7786e-16   -4.32963e-8     -0.000173639   -0.000171662
  1.00219e-13   2.05097e-10   3.80035e-16  -1.28633e-7  …   0.000207556    0.000205176
  1.26036e-13   2.57934e-10   5.21369e-16  -1.61765e-7     -0.00024099    -0.000238237
  ⋮                                                    ⋱
 -8.2267e-14   -1.69652e-10  -4.13628e-16   1.06245e-7  …  -0.000206971    0.000204996
 -1.35746e-13  -2.79302e-10  -3.22874e-16   1.74757e-7      0.000173229   -0.000171501
 -5.56245e-14  -1.14643e-10  -2.56398e-16   7.17457e-8     -0.000139038    0.000137651
 -3.09462e-15  -6.62183e-12  -1.87375e-16   4.14504e-9      0.000104581   -0.000103493
  1.88641e-13   3.87195e-10  -1.47332e-16  -2.42136e-7     -6.98431e-5     6.91219e-5
  9.59004e-14   1.9684e-10   -7.15417e-17  -1.23101e-7  …   3.48915e-5    -3.46147e-5
```

```
eb = eigenvectors(Hb, 800)
```

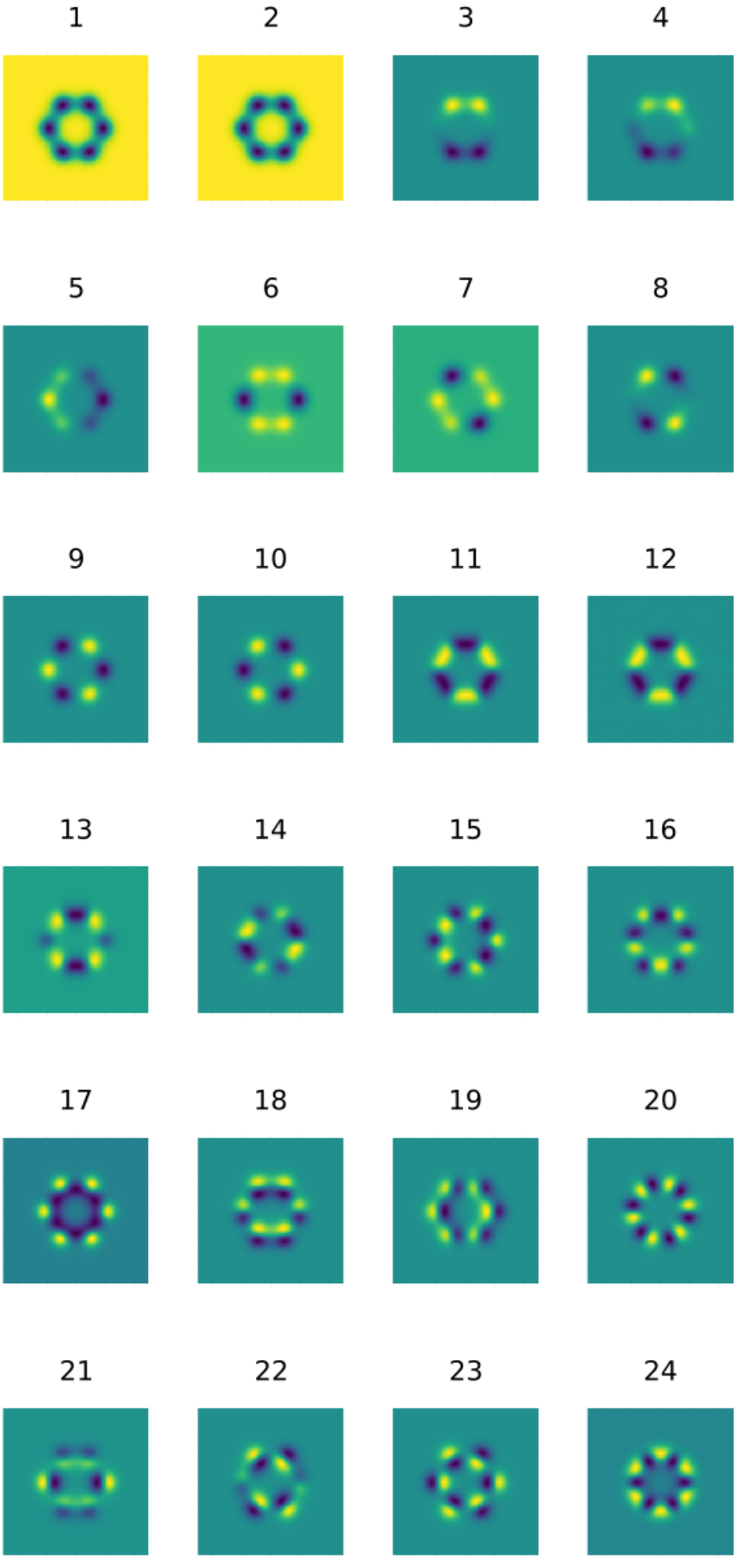
```
erb = 201×201×800 Array{Float64, 3}:
[:, :, 1] =
 -1.58931e-13  -2.65439e-13  -3.5434e-13   …  -2.21505e-13   1.61188e-14
 -3.36586e-13  -5.29175e-13  -5.2567e-13      -2.87811e-13  -4.17064e-15
 -1.65084e-13  -4.89236e-13  -4.95438e-13     -6.63927e-14  -9.95909e-15
 -1.28191e-13  -3.13136e-13  -1.89828e-13     -4.41619e-14   4.27922e-14
  3.38185e-14   1.2481e-13    2.45244e-13      2.762e-13     1.04192e-13
  1.00219e-13   3.71894e-13   6.70309e-13  …   2.89017e-13   3.04603e-13
  1.26036e-13   3.64352e-13   8.32389e-13      3.6664e-13    2.80195e-13
  ⋮                                      ⋱                ⋮
  1.32847e-14  -8.0932e-14   -2.02212e-14  …   2.32059e-14  -8.2267e-14
 -7.2207e-14   -2.32585e-13  -4.02648e-13     -4.65203e-14  -1.35746e-13
 -1.60402e-13  -2.52358e-13  -4.73759e-13     -3.01343e-14  -5.56245e-14
 -2.64119e-13  -3.91169e-13  -4.99462e-13      1.25914e-13  -3.09462e-15
 -2.72143e-13  -4.75648e-13  -7.19794e-13      1.93017e-13   1.88641e-13
 -2.04792e-13  -5.14981e-13  -4.8976e-13   …  -4.28106e-14   9.59004e-14

[:, :, 2] =
 -3.26475e-10  -5.45329e-10  -7.28008e-10  …  -4.55075e-10   3.29632e-11
 -6.91442e-10  -1.08716e-9   -1.08012e-9      -5.91473e-10  -8.7956e-12
 -3.39339e-10  -1.0053e-9    -1.01832e-9      -1.3694e-10   -2.07462e-11
 -2.63668e-10  -6.43902e-10  -3.91116e-10     -9.14699e-11   8.74948e-11
  6.88897e-11   2.5513e-10    5.01854e-10      5.66129e-10   2.13456e-10
  2.05097e-10   7.62196e-10   1.37421e-9   …   5.92097e-10   6.24835e-10
  2.57934e-10   7.46367e-10   1.70644e-9       7.51099e-10   5.74521e-10
  ⋮                                      ⋱                ⋮
  2.65568e-11  -1.67671e-10  -4.38344e-11  …   4.62228e-11  -1.69652e-10
 -1.48871e-10  -4.78816e-10  -8.28734e-10     -9.66239e-11  -2.79302e-10
 -3.2983e-10   -5.19144e-10  -9.74345e-10     -6.2715e-11   -1.14643e-10
 -5.42716e-10  -8.03916e-10  -1.02664e-9       2.57987e-10  -6.62183e-12
 -5.5909e-10   -9.77227e-10  -1.47879e-9       3.95984e-10   3.87195e-10
 -4.20653e-10  -1.05779e-9   -1.00608e-9   …  -8.81381e-11   1.9684e-10
```

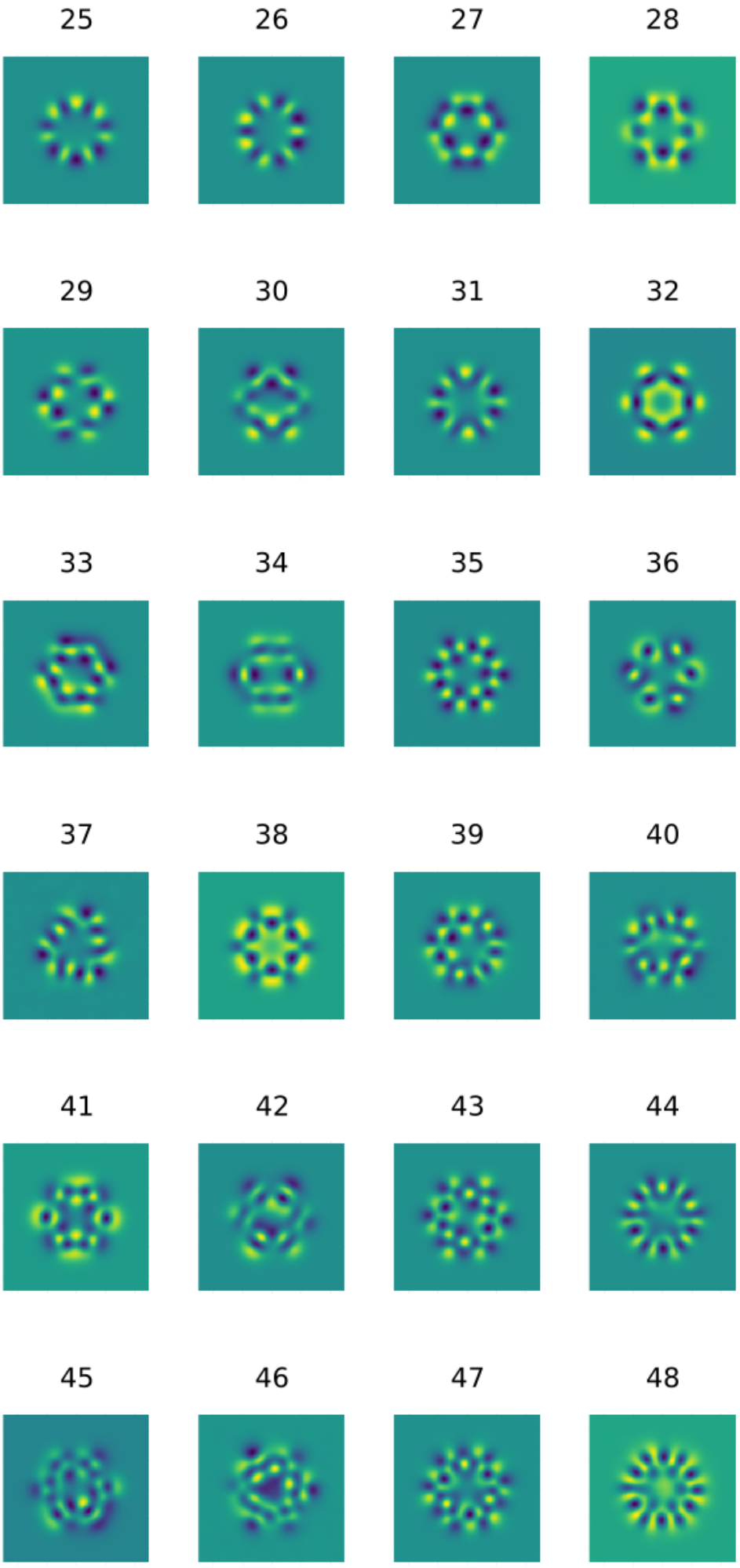
```
1 erb = reshape(eb, N, N, :)
```

1




```
with(ratio=1, title="vlastní vektor $(kb)", size=(600,630)) do
    heatmap(x, y, erb[:,:,kb],
        xlabel="x [nm]", ylabel="y [nm]")
end
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

```
with(ratio=1, size=(600,2500), plot_title="", legend=false, showaxis=false) do
    plotsb = map(1:48) do k
        heatmap(x, y, erb[:,:,k], title="$k")
    end
    plot(plotsb..., layout=(12,4))
end
```