```
PlotlyBackend()
```

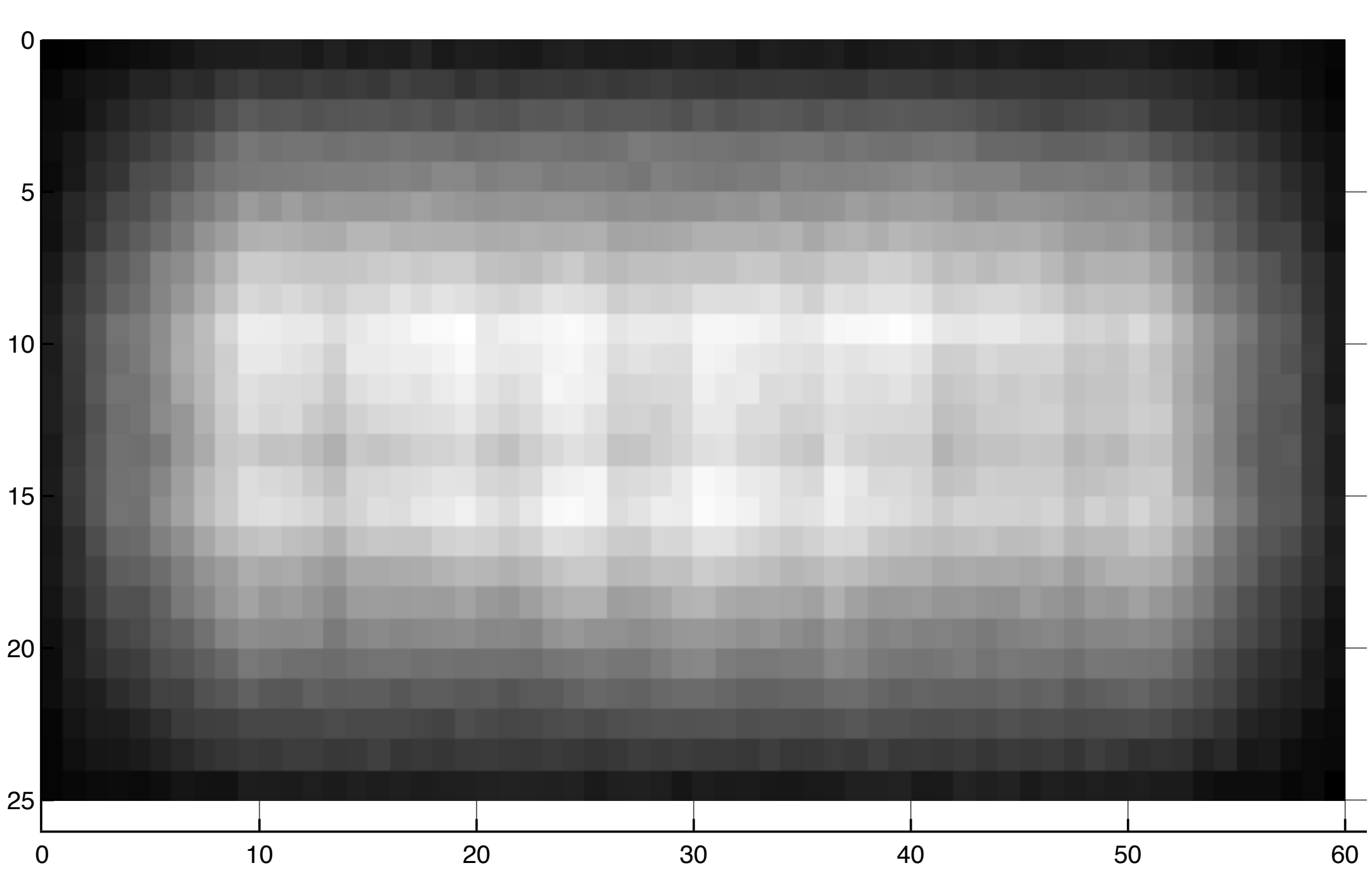
```julia
begin
    using NPZ ✓
    using LinearAlgebra ✓
    using Images ✓
    using Plots ✓
    plotly()
end
```

590 μs

Загрузка данных

```julia
begin
    local data = npzread("./data.npz")
    A = data["A"]
    C = data["C"]
end;
```

Построим изображение после свёртки:



```julia
plot(Gray.(A ./ maximum(A)))
```

Определим функции, разворачивающие матрицу в вектор и обратно:

```
mat2vec (generic function with 1 method)
```

```julia
function mat2vec(A)
    A = @view A[end:-1:begin, :]
    return vec(copy(A'))
end
```

```
vec2mat (generic function with 2 methods)
```

```julia
function vec2mat(a, shape = (16, 51))
    A = reshape(a, reverse(shape))'
    return A[end:-1:begin, :]
end
```

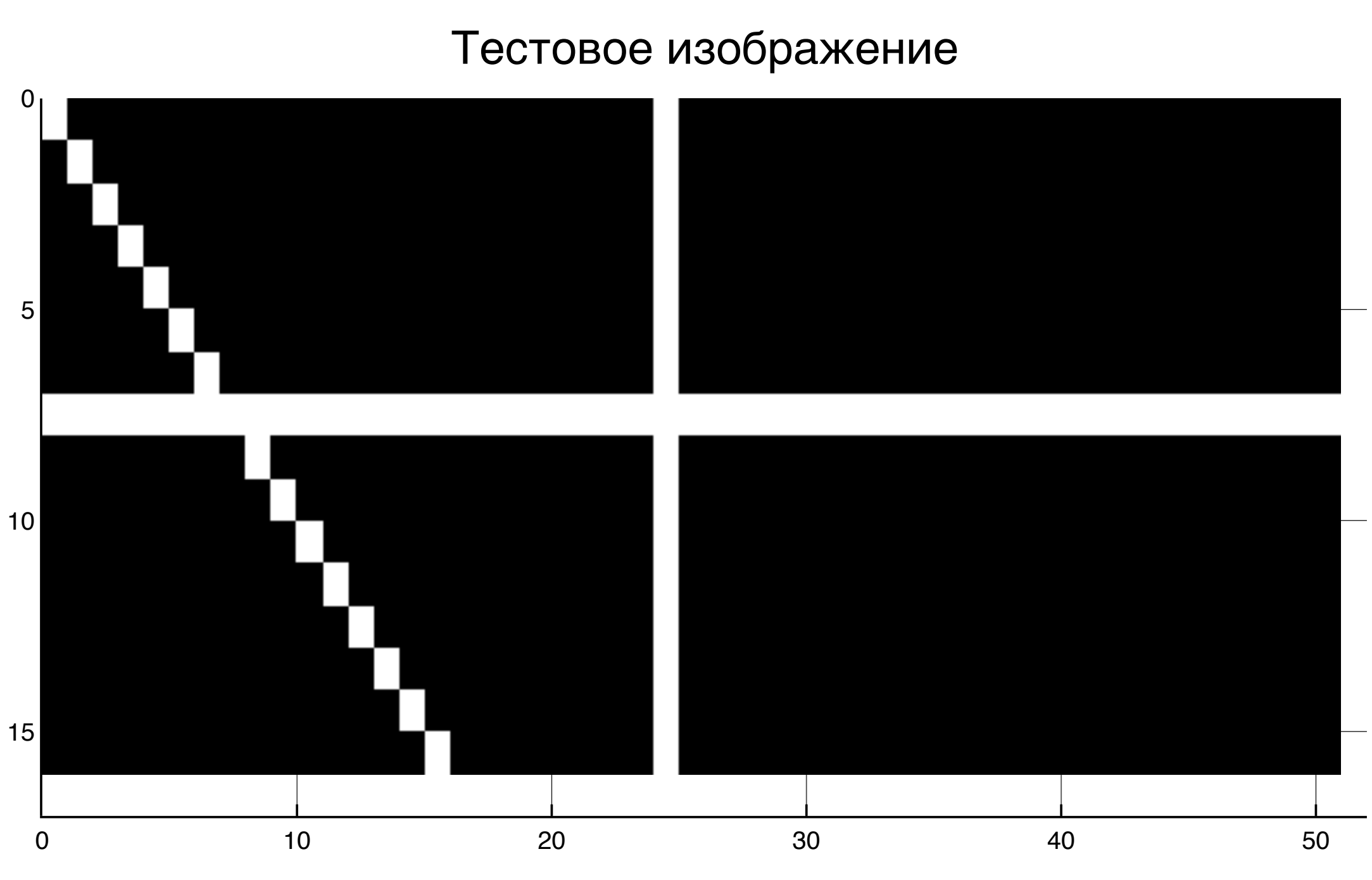
```
a =
▸[17.5041, 27.7684, 37.7167, 41.8572, 35.2522, 50.4719, 72.9888, 66.4267, 63.3316, 102.69
```

```julia
a = mat2vec(A)
```

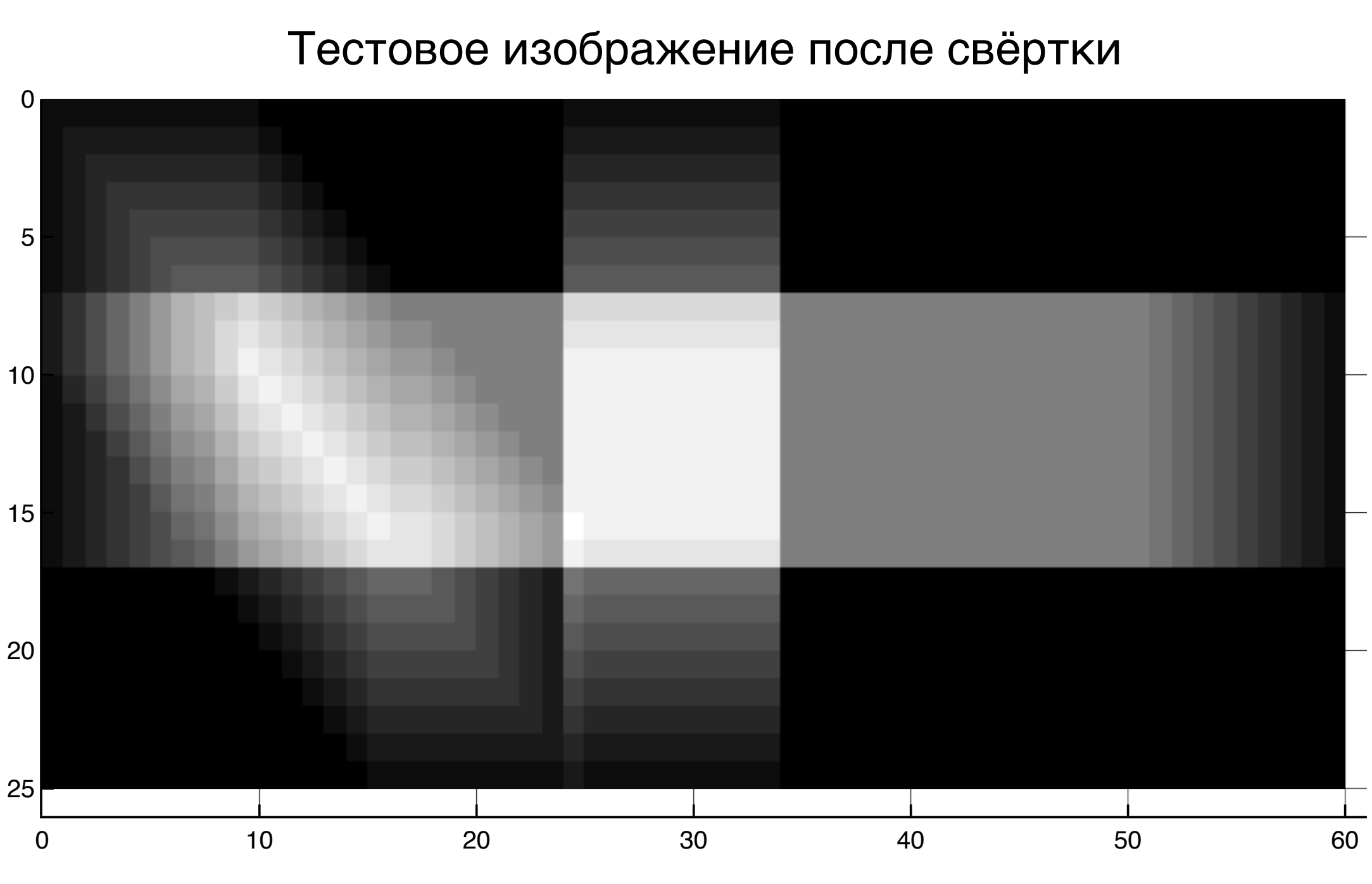
Тест работы фильтра

Чтобы понять, как работает фильтр, создадим тестовое изображение и посмотрим, как оно выглядит после свёртки:

```julia
begin
    B = zeros(16, 51)
    B[8, :]  .= 1
    B[:, 25] .= 1
    for i ∈ 1 : minimum(size(B))
        B[i, i] = 1
    end
    b = mat2vec(B)
end;
```



```julia
plot(Gray.(B ./ maximum(B)), title = "Тестовое изображение", topmargin = 20Plots.px)
```



```julia
plot_vec(C * b, (25, 60), title = "Тестовое изображение после свёртки", topmargin =
20Plots.px)
```

Вывод:

Каждый светлый пиксель под действием C заменяется на прямоугольник.

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

```julia
function plot_vec(a, shape = (16,51); kwargs...)
    # Вспомогательная функция, которая разворачивает вектор
    # в матрицу и рисует его
    min, max = extrema(a)
    a .-= min
    a ./= (max - min)
    A = vec2mat(a, shape)
    return plot(Gray.(A); kwargs...)
end
```

Обращение фильтра

```
singular values:
816-element Vector{Float64}:
 3.581791498948593
 3.441349727991941
 3.215493756964926
 2.9160604144673874
 2.645571112047677
 2.5587103838645855
 2.5418384709163333
 ⋮
 0.006237570730896662
 0.006197897718499658
 0.0059139578558154
 0.005653575076996135
 0.005617616470065084
 0.005360260617876323
Vt factor:
816×816 Matrix{Float64}:
 -0.00539188  -0.00668     -0.00804174  …  -0.00804174  -0.00668     -0.00539188
 -0.0107767   -0.0131548   -0.0155647       0.0155647    0.0131548    0.0107767
  0.0161495    0.0192163    0.0220591       0.0220591    0.0192163    0.0161495
 -0.0215095   -0.02465     -0.0270315       0.0270315    0.02465      0.0215095
  0.0104372    0.0129306    0.0155666      -0.0155666   -0.0129306   -0.0104372
 -0.0268648   -0.0292298   -0.0300082   …  -0.0300082   -0.0292298   -0.0268648
  0.0208608    0.025464     0.0301289       0.0301289    0.025464     0.0208608
  ⋮                                     ⋱                             ⋮
 -0.0129353    0.0219171   -0.0103685   …  -0.0103685    0.0219171   -0.0129353
 -0.00492555   0.0163947   -0.0254342       0.0254342   -0.0163947    0.00492555
  0.00794542  -0.0168084    0.018621        0.018621    -0.0168084    0.00794542
 -0.00677787   0.0114842   -0.00543294      0.00543294  -0.0114842    0.00677787
 -0.00258091   0.00859056  -0.0133271      -0.0133271    0.00859056  -0.00258091
  0.00416327  -0.00880733   0.00975709  …  -0.00975709   0.00880733  -0.00416327
```

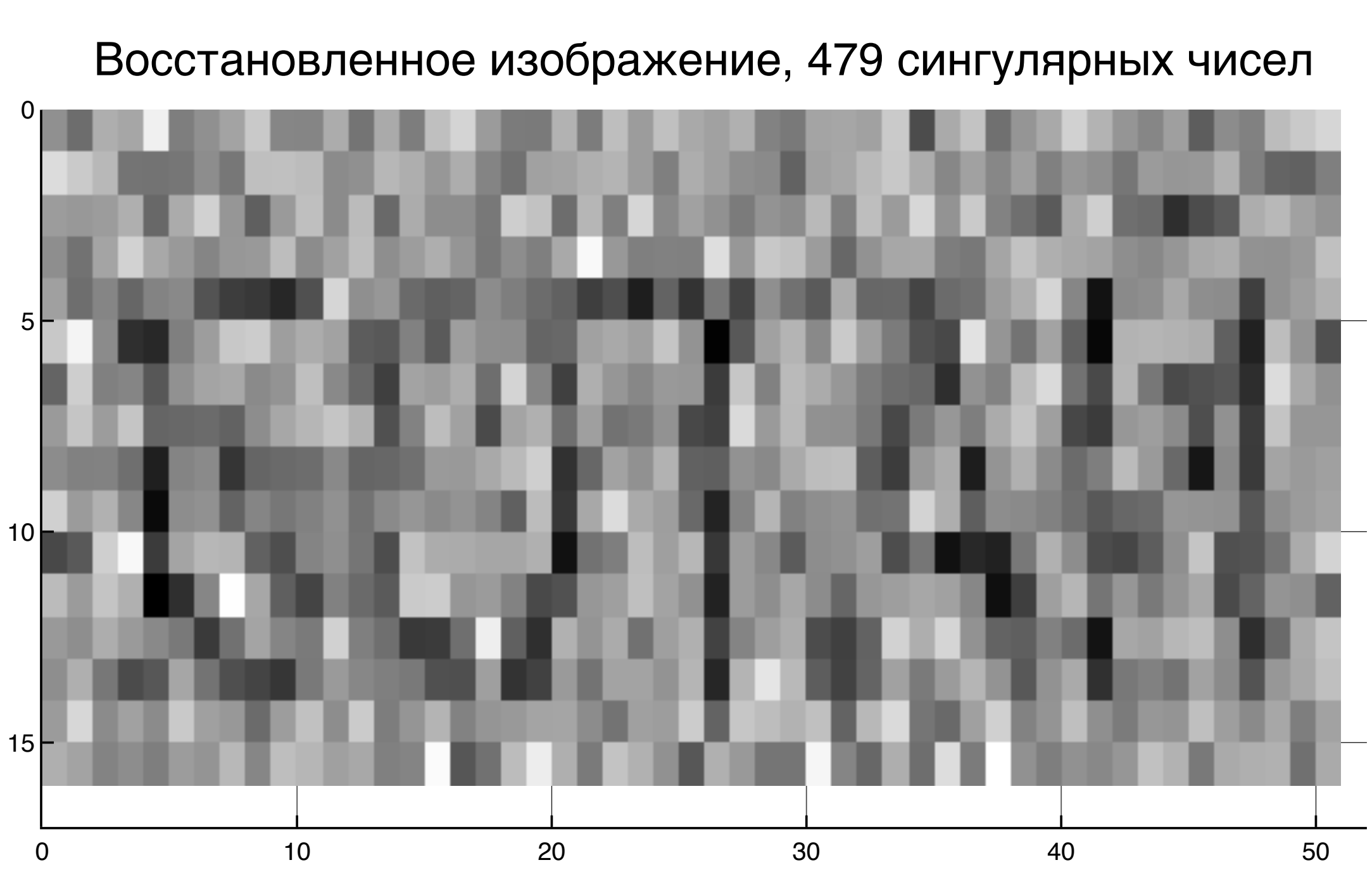
```julia
C_SVD = svd(C)
```

```julia
plot_vec(C_SVD \ a, title = "Восстановленное изображение; все сингулярные числа",
topmargin = 20Plots.px)
```

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

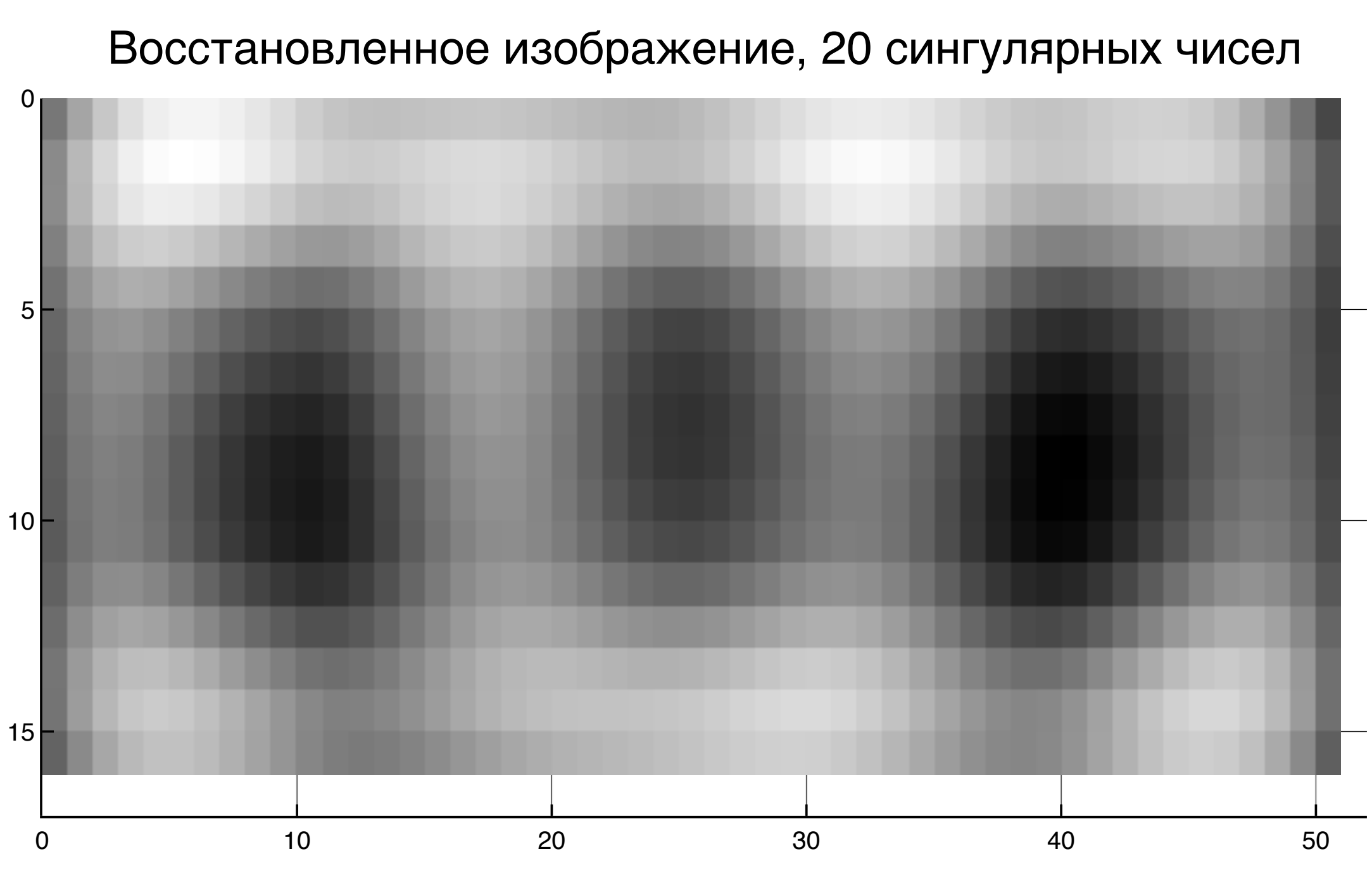
```julia
function approx_lsolve(n_svalues, a = a, C_SVD = C_SVD)
    # Функция, которая решает недо- или переопределённую систему
    # с помощью SVD.
    # Используются только первые n_svalues сингулярных чисел.
    U, S, V = C_SVD
    U = U[:, 1:n_svalues]
    S = S[1:n_svalues]
    V = V[:, 1:n_svalues]
    C_SVD = SVD(U, S, V')
    return C_SVD \ a
end
```

Попробуем использовать только часть сингулярных чисел:



```julia
plot_vec(approx_lsolve(479), title = "Восстановленное изображение, 479 сингулярных
чисел", topmargin = 20Plots.px)
```

```julia
plot_vec(approx_lsolve(125), title = "Восстановленное изображение, 125 сингулярных
чисел", topmargin = 20Plots.px)
```

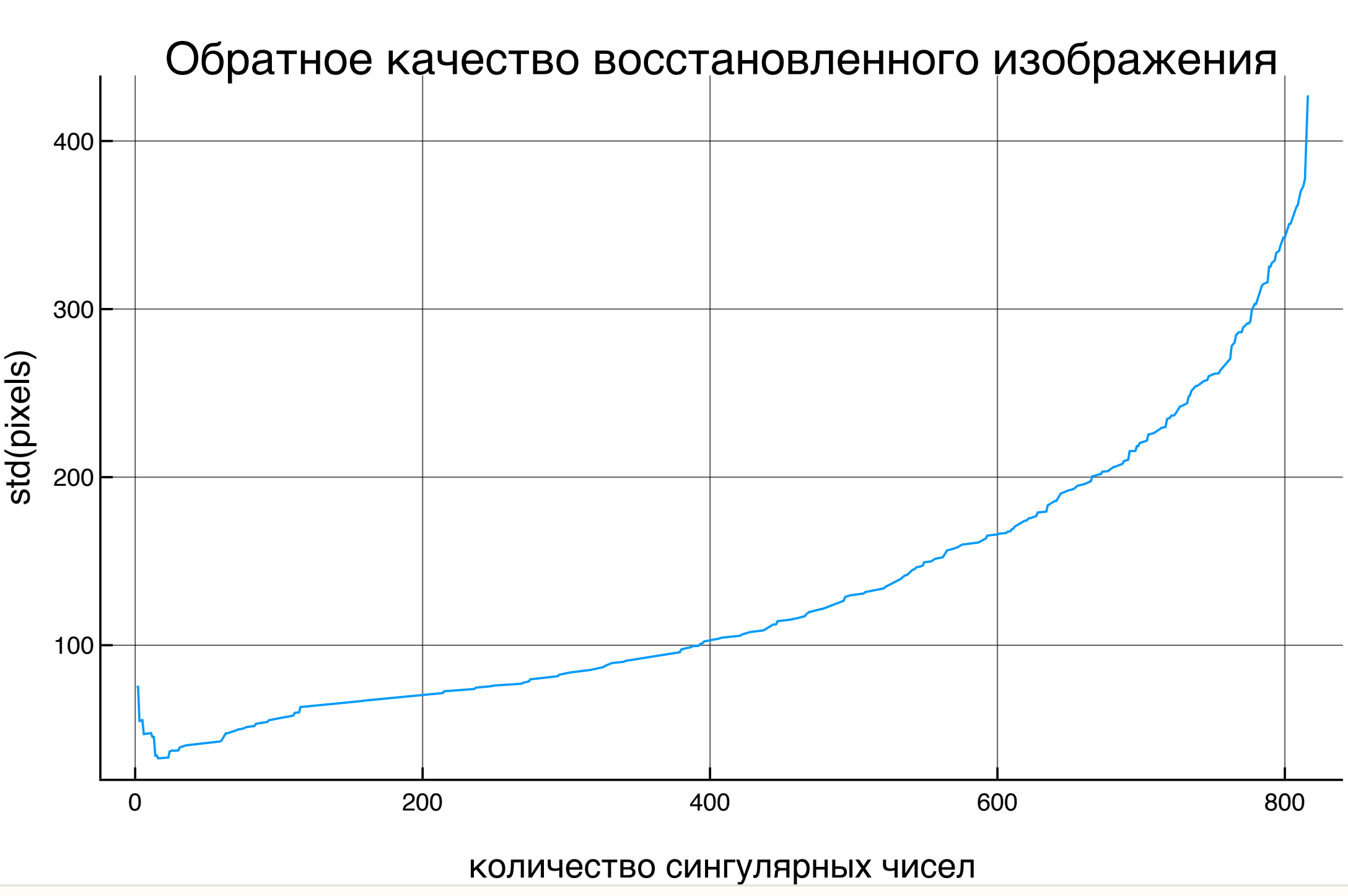


Настоящее изображение, вероятно, не похоже на белый шум и содержит большие равномерно окрашенные участки.

Я перебрал различные количества сингулярных чисел и выбрал то, при котором стандартное отклонение между пикселями получается наименьшим.

Это происходит при использовании 20 сингулярных чисел.

```julia
begin
    image_qualities = zeros(size(C_SVD.S))
    for i in eachindex(image_qualities)
        restored_vec = approx_lsolve(i)
        image_qualities[i] = std(restored_vec)
    end
end
```



```julia
plot(image_qualities, title = "Обратное качество восстановленного изображения",
xlabel = "количество сингулярных чисел", ylabel = "std(pixels)", legend = false)
```

📚 Live docs

FAQ

How can we make Pluto.jl better? Instant feedback... Send