Solving Stiff Equations

Chris Rackauckas

September 3, 2021

This tutorial is for getting into the extra features for solving stiff ordinary differential equations in an efficient manner. Solving stiff ordinary differential equations requires specializing the linear solver on properties of the Jacobian in order to cut down on the O(n^3) linear solve and the O(n^2) back-solves. Note that these same functions and controls also extend to stiff SDEs, DDEs, DAEs, etc.

0.1 Code Optimization for Differential Equations

0.1.1 Writing Efficient Code

For a detailed tutorial on how to optimize one's DifferentialEquations.jl code, please see the Optimizing DiffEq Code tutorial.

0.1.2 Choosing a Good Solver

Choosing a good solver is required for getting top notch speed. General recommendations can be found on the solver page (for example, the ODE Solver Recommendations). The current recommendations can be simplified to a Rosenbrock method (`Rosenbrock23` or `Rodas5`) for smaller (<50 ODEs) problems, ESDIRK methods for slightly larger (`TRBDF2` or `KenCarp4` for <2000 ODEs), and Sundials `CVODE_BDF` for even larger problems. `lsoda` from LSODA.jl is generally worth a try.

More details on the solver to choose can be found by benchmarking. See the DiffEqBenchmarks to compare many solvers on many problems.

0.1.3 Check Out the Speed FAQ

See this FAQ for information on common pitfalls and how to improve performance.

0.1.4 Setting Up Your Julia Installation for Speed

Julia uses an underlying BLAS implementation for its matrix multiplications and factorizations. This library is automatically multithreaded and accelerates the internal linear algebra of DifferentialEquations.jl. However, for optimality, you should make sure that the number of BLAS threads that you are using matches the number of physical cores and not the number of logical cores. See this issue for more details.

To check the number of BLAS threads, use:

```
ccall((:openblas_get_num_threads64_, Base.libblas_name), Cint, ())
```

```
4
```

If I want to set this directly to 4 threads, I would use:

```
using LinearAlgebra
LinearAlgebra.BLAS.set_num_threads(4)
```

Additionally, in some cases Intel's MKL might be a faster BLAS than the standard BLAS that ships with Julia (OpenBLAS). To switch your BLAS implementation, you can use MKL.jl which will accelerate the linear algebra routines. Please see the package for the limitations.

0.1.5 Use Accelerator Hardware

When possible, use GPUs. If your ODE system is small and you need to solve it with very many different parameters, see the ensembles interface and DiffEqGPU.jl. If your problem is large, consider using a CuArray for the state to allow for GPU-parallelism of the internal linear algebra.

0.2 Speeding Up Jacobian Calculations

When one is using an implicit or semi-implicit differential equation solver, the Jacobian must be built at many iterations and this can be one of the most expensive steps. There are two pieces that must be optimized in order to reach maximal efficiency when solving stiff equations: the sparsity pattern and the construction of the Jacobian. The construction is filling the matrix `J` with values, while the sparsity pattern is what `J` to use.

The sparsity pattern is given by a prototype matrix, the `jac_prototype`, which will be copied to be used as `J`. The default is for `J` to be a `Matrix`, i.e. a dense matrix. However, if you know the sparsity of your problem, then you can pass a different matrix type. For example, a `SparseMatrixCSC` will give a sparse matrix. Additionally, structured matrix types like `Tridiagonal`, `BandedMatrix` (from BandedMatrices.jl), `BlockBandedMatrix` (from BlockBandedMatrices.jl), and more can be given. DifferentialEquations.jl will internally use this matrix type, making the factorizations faster by utilizing the specialized forms.

For the construction, there are 3 ways to fill `J`:

- The default, which uses normal finite/automatic differentiation
- A function `jac(J,u,p,t)` which directly computes the values of `J`
- A `colorvec` which defines a sparse differentiation scheme.

We will now showcase how to make use of this functionality with growing complexity.

0.2.1 Declaring Jacobian Functions

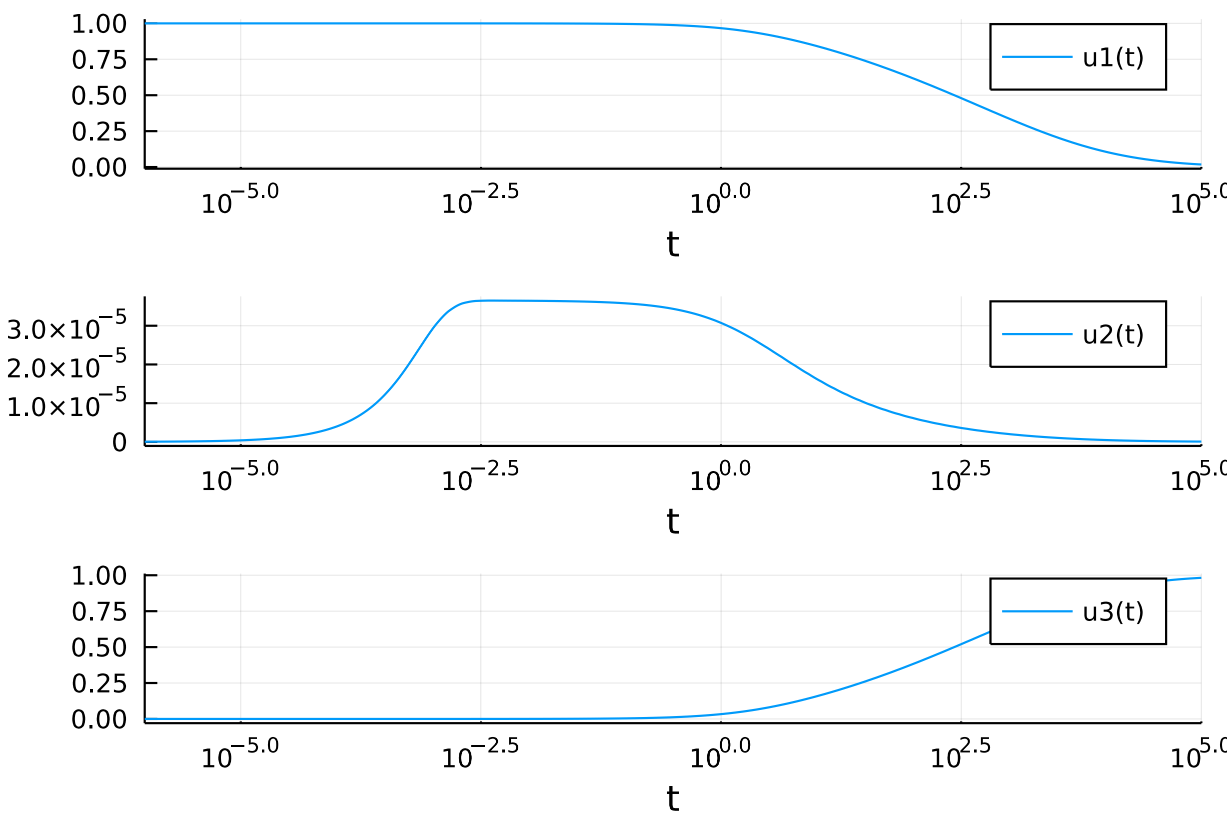
Let's solve the Rosenbrock equations:

$$
\begin{align}
dy_1 &= -0.04y\_1 + 10^4 y_2 y_3 \tag{1}\\
dy_2 &= 0.04y_1 - 10^4 y_2 y_3 - 3 * 10^7 y_2^2 \tag{2}\\
dy_3 &= 3 * 10^7 y_3^2 \tag{3}\\
& \tag{4}
\end{align}
$$

In order to reduce the Jacobian construction cost, one can describe a Jacobian function by using the `jac` argument for the `ODEFunction`. First, let's do a standard `ODEProblem`:

```
using DifferentialEquations
function rober(du,u,p,t)
  y_1,y_2,y_3 = u
  k_1,k_2,k_3 = p
  du[1] = -k_1*y_1+k_3*y_2*y_3
  du[2] =  k_1*y_1-k_2*y_2^2-k_3*y_2*y_3
  du[3] =  k_2*y_2^2
  nothing
end
prob = ODEProblem(rober,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
sol = solve(prob,Rosenbrock23())

using Plots
plot(sol, xscale=:log10, tspan=(1e-6, 1e5), layout=(3,1))
```




```
using BenchmarkTools
@btime solve(prob)
```

```
580.174 μs (2561 allocations: 186.89 KiB)
retcode: Success
Interpolation: automatic order switching interpolation
t: 115-element Vector{Float64}:
      0.0
      0.0014148468219250373
      0.0020449182545311173
      0.0031082402716566307
      0.004077787050059496
      0.005515332443361059
      0.007190040962774541
      0.009125372578778032
      0.011053912492732977
      0.012779077276958607
      ⋮
  47335.55742427336
  52732.00629853751
  58693.72275675742
  65277.99247104326
  72548.193682209
  80574.55524404174
  89435.04313420167
  99216.40190401232
 100000.0
u: 115-element Vector{Vector{Float64}}:
 [1.0, 0.0, 0.0]
 [0.9999434113193613, 3.283958829839966e-5, 2.374909234028646e-5]
 [0.9999182177783585, 3.5542680136344576e-5, 4.6239541505020636e-5]
 [0.999875715036629, 3.630246933484973e-5, 8.798249403609502e-5]
 [0.9998369766077329, 3.646280308115454e-5, 0.00012656058918590176]
 [0.9997795672444667, 3.6466430856422276e-5, 0.00018396632467683696]
 [0.9997127287139348, 3.644727999289594e-5, 0.0002508240060722832]
 [0.9996355450022019, 3.6366816179962554e-5, 0.0003280881816181881]
 [0.9995586925734838, 3.6018927453310745e-5, 0.00040528849906290245]
 [0.9994899965196854, 3.468694637784628e-5, 0.00047531653393682193]
 ⋮
 [0.03394368533138813, 1.4047985954839008e-7, 0.9660561741887524]
 [0.031028978802635446, 1.280360882615162e-7, 0.9689708931612764]
 [0.02835436649772506, 1.1668210763639173e-7, 0.971645516820168]
 [0.02590132862868119, 1.0632277796078e-7, 0.9740985650485414]
 [0.023652547707489525, 9.687113505658095e-8, 0.9763473554213756]
 [0.021591864255513585, 8.824768851310993e-8, 0.9784080474967981]
 [0.01970422745458613, 8.037977845291491e-8, 0.9802956921656356]
 [0.017975643191251073, 7.320098956514714e-8, 0.9820242836077591]
 [0.01785056623499974, 7.26838436117136e-8, 0.9821493610811564]
```

Now we want to add the Jacobian. First we have to derive the Jacobian $\frac{df_i}{du_j}$ which is `J[i,j]`. From this we get:

```
function rober_jac(J,u,p,t)
  y_1,y_2,y_3 = u
  k_1,k_2,k_3 = p
  J[1,1] = k_1 * -1
  J[2,1] = k_1
  J[3,1] = 0
  J[1,2] = y_3 * k_3
  J[2,2] = y_2 * k_2 * -2 + y_3 * k_3 * -1
  J[3,2] = y_2 * 2 * k_2
```

```
  J[1,3] = k_3 * y_2
  J[2,3] = k_3 * y_2 * -1
  J[3,3] = 0
  nothing
end
f = ODEFunction(rober, jac=rober_jac)
prob_jac = ODEProblem(f,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))

@btime solve(prob_jac)
```

```
360.706 μs (2002 allocations: 126.28 KiB)
retcode: Success
Interpolation: automatic order switching interpolation
t: 115-element Vector{Float64}:
      0.0
      0.0014148468219250373
      0.0020449182545311173
      0.0031082402716566307
      0.004077787050059496
      0.005515332443361059
      0.007190040962774541
      0.009125372578778032
      0.011053912492732977
      0.012779077276958607
      ⋮
  45964.060340548356
  51219.40381376205
  57025.01899700374
  63436.021374561584
  70513.1073617524
  78323.14229130604
  86939.82338876331
  96444.41085674686
 100000.0
u: 115-element Vector{Vector{Float64}}:
 [1.0, 0.0, 0.0]
 [0.9999434113193613, 3.283958829839966e-5, 2.374909234028646e-5]
 [0.9999182177783585, 3.5542680136344576e-5, 4.6239541505020636e-5]
 [0.999875715036629, 3.630246933484973e-5, 8.798249403609502e-5]
 [0.9998369766077329, 3.646280308115454e-5, 0.00012656058918590176]
 [0.9997795672444667, 3.6466430856422276e-5, 0.00018396632467683696]
 [0.9997127287139348, 3.644727999289594e-5, 0.0002508240060722832]
 [0.9996355450022019, 3.6366816179962554e-5, 0.0003280881816181881]
 [0.9995586925734838, 3.6018927453310745e-5, 0.00040528849906290245]
 [0.9994899965196854, 3.468694637784628e-5, 0.00047531653393682193]
 ⋮
 [0.03478048133177493, 1.4406682005231008e-7, 0.9652193746014031]
 [0.03179591062189176, 1.313038656880417e-7, 0.9682039580742408]
 [0.029057356622057315, 1.1966100432939363e-7, 0.9709425237169371]
 [0.02654597011713668, 1.0904070990251299e-7, 0.9734539208421517]
 [0.024244118287194777, 9.935385522693504e-8, 0.9757557823589477]
 [0.022135344621501105, 9.05190025093182e-8, 0.9778645648594945]
 [0.02020432071854, 8.246174295748071e-8, 0.9797955968197154]
 [0.018436796681356796, 7.511410189106845e-8, 0.9815631282045397]
 [0.01785426048218692, 7.269900678199638e-8, 0.9821456668188047]
```

0.2.2 Automatic Derivation of Jacobian Functions

But that was hard! If you want to take the symbolic Jacobian of numerical code, we can make use of ModelingToolkit.jl to symbolicify the numerical code and do the symbolic calculation and return the Julia code for this.

```
using ModelingToolkit
de = modelingtoolkitize(prob)
ModelingToolkit.generate_jacobian(de)[2] # Second is in-place
```

```
:(function (var"##out#6147", var"##arg#6145", var"##arg#6146", t)
      #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-bc97
-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:282 =#
      #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-bc97
-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:283 =#
      let var"x_1(t)" = #= /root/.cache/julia-buildkite-plugin/depots/a6029d
3a-f78b-41ea-bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:169
 =# @inbounds(var"##arg#6145"[1]), var"x_2(t)" = #= /root/.cache/julia-build
kite-plugin/depots/a6029d3a-f78b-41ea-bc97-28aa57c6c6ea/packages/SymbolicUt
ils/9iQGH/src/code.jl:169 =# @inbounds(var"##arg#6145"[2]), var"x_3(t)" = #=
 /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-bc97-28aa57c
6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:169 =# @inbounds(var"##arg#6
145"[3]), α_1 = #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-
41ea-bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:169 =# @inb
ounds(var"##arg#6146"[1]), α_2 = #= /root/.cache/julia-buildkite-plugin/depo
ts/a6029d3a-f78b-41ea-bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/co
de.jl:169 =# @inbounds(var"##arg#6146"[2]), α_3 = #= /root/.cache/julia-buil
dkite-plugin/depots/a6029d3a-f78b-41ea-bc97-28aa57c6c6ea/packages/SymbolicU
tils/9iQGH/src/code.jl:169 =# @inbounds(var"##arg#6146"[3])
          #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-
bc97-28aa57c6c6ea/packages/Symbolics/h8kPL/src/build_function.jl:331 =#
          #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-
bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:329 =# @inbounds
 begin
                  #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f7
8b-41ea-bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:325 =#
                  var"##out#6147"[1] = (*)(-1, α_1)
                  var"##out#6147"[2] = α_1
                  var"##out#6147"[3] = 0
                  var"##out#6147"[4] = (*)(α_3, var"x_3(t)")
                  var"##out#6147"[5] = (+)((*)(-2, α_2, var"x_2(t)"), (*)(-1,
 α_3, var"x_3(t)"))
                  var"##out#6147"[6] = (*)(2, α_2, var"x_2(t)")
                  var"##out#6147"[7] = (*)(α_3, var"x_2(t)")
                  var"##out#6147"[8] = (*)(-1, α_3, var"x_2(t)")
                  var"##out#6147"[9] = 0
                  #= /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f7
8b-41ea-bc97-28aa57c6c6ea/packages/SymbolicUtils/9iQGH/src/code.jl:327 =#
                  nothing
              end
      end
  end)
```

which outputs:

```
:((##MTIIPVar#376, u, p, t)->begin
          #= C:\Users\accou\.julia\packages\ModelingToolkit\czHtj\src\utils.jl:65 =#
          #= C:\Users\accou\.julia\packages\ModelingToolkit\czHtj\src\utils.jl:66 =#
          let (x_1, x_2, x_3, α_1, α_2, α_3) = (u[1], u[2], u[3], p[1], p[2], p[3])
              ##MTIIPVar#376[1] = α_1 * -1
```

```
            ##MTIIPVar#376[2] = α_1
            ##MTIIPVar#376[3] = 0
            ##MTIIPVar#376[4] = x_3 * α_3
            ##MTIIPVar#376[5] = x_2 * α_2 * -2 + x_3 * α_3 * -1
            ##MTIIPVar#376[6] = x_2 * 2 * α_2
            ##MTIIPVar#376[7] = α_3 * x_2
            ##MTIIPVar#376[8] = α_3 * x_2 * -1
            ##MTIIPVar#376[9] = 0
        end
        #= C:\Users\accou\.julia\packages\ModelingToolkit\czHtj\src\utils.jl:67 =#
        nothing
    end)
```

Now let's use that to give the analytical solution Jacobian:

```
jac = eval(ModelingToolkit.generate_jacobian(de)[2])
f = ODEFunction(rober, jac=jac)
prob_jac = ODEProblem(f,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
```

```
ODEProblem with uType Vector{Float64} and tType Float64. In-place: true
timespan: (0.0, 100000.0)
u0: 3-element Vector{Float64}:
 1.0
 0.0
 0.0
```

0.2.3 Declaring a Sparse Jacobian

Jacobian sparsity is declared by the `jac_prototype` argument in the `ODEFunction`. Note that you should only do this if the sparsity is high, for example, 0.1% of the matrix is non-zeros, otherwise the overhead of sparse matrices can be higher than the gains from sparse differentiation!

But as a demonstration, let's build a sparse matrix for the Rober problem. We can do this by gathering the `I` and `J` pairs for the non-zero components, like:

```
I = [1,2,1,2,3,1,2]
J = [1,1,2,2,2,3,3]
using SparseArrays
jac_prototype = sparse(I,J,1.0)
```

```
3×3 SparseArrays.SparseMatrixCSC{Float64, Int64} with 7 stored entries:
 1.0  1.0  1.0
 1.0  1.0  1.0
  ⋅   1.0   ⋅
```

Now this is the sparse matrix prototype that we want to use in our solver, which we then pass like:

```
f = ODEFunction(rober, jac=jac, jac_prototype=jac_prototype)
prob_jac = ODEProblem(f,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
```

```
ODEProblem with uType Vector{Float64} and tType Float64. In-place: true
timespan: (0.0, 100000.0)
u0: 3-element Vector{Float64}:
 1.0
 0.0
 0.0
```

0.2.4 Automatic Sparsity Detection

One of the useful companion tools for DifferentialEquations.jl is SparsityDetection.jl. This allows for automatic declaration of Jacobian sparsity types. To see this in action, let's look at the 2-dimensional Brusselator equation:

```
const N = 32
const xyd_brusselator = range(0,stop=1,length=N)
brusselator_f(x, y, t) = (((x-0.3)^2 + (y-0.6)^2) <= 0.1^2) * (t >= 1.1) * 5.
limit(a, N) = a == N+1 ? 1 : a == 0 ? N : a
function brusselator_2d_loop(du, u, p, t)
  A, B, alpha, dx = p
  alpha = alpha/dx^2
  @inbounds for I in CartesianIndices((N, N))
    i, j = Tuple(I)
    x, y = xyd_brusselator[I[1]], xyd_brusselator[I[2]]
    ip1, im1, jp1, jm1 = limit(i+1, N), limit(i-1, N), limit(j+1, N), limit(j-1, N)
    du[i,j,1] = alpha*(u[im1,j,1] + u[ip1,j,1] + u[i,jp1,1] + u[i,jm1,1] - 4u[i,j,1]) +
                B + u[i,j,1]^2*u[i,j,2] - (A + 1)*u[i,j,1] + brusselator_f(x, y, t)
    du[i,j,2] = alpha*(u[im1,j,2] + u[ip1,j,2] + u[i,jp1,2] + u[i,jm1,2] - 4u[i,j,2]) +
                A*u[i,j,1] - u[i,j,1]^2*u[i,j,2]
    end
end
p = (3.4, 1., 10., step(xyd_brusselator))
```

```
(3.4, 1.0, 10.0, 0.03225806451612903)
```

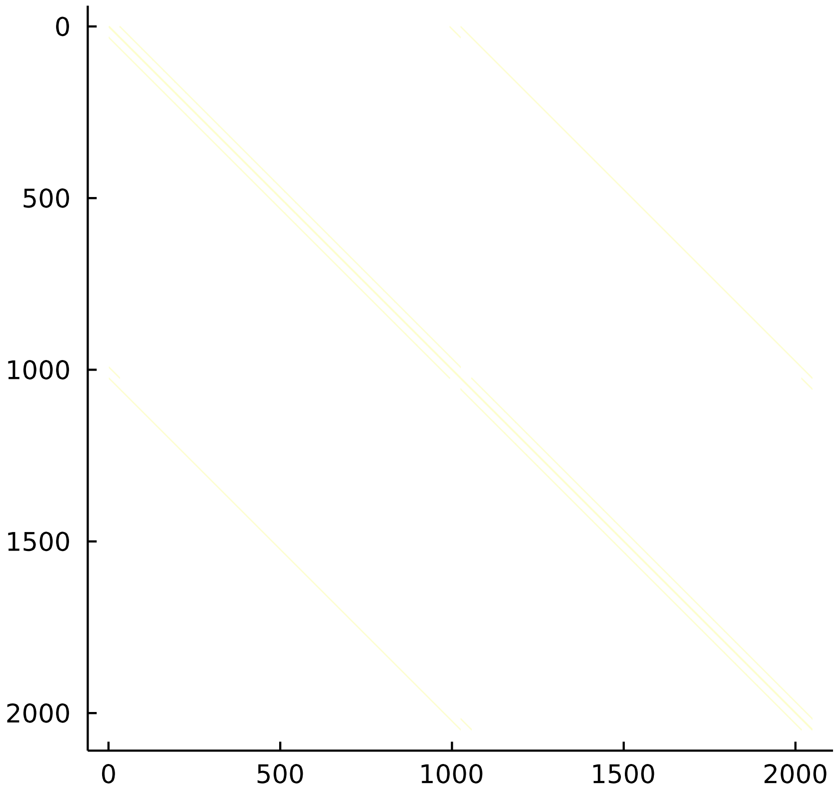
Given this setup, we can give and example `input` and `output` and call `sparsity!` on our function with the example arguments and it will kick out a sparse matrix with our pattern, that we can turn into our `jac_prototype`.

```
using SparsityDetection, SparseArrays
input = rand(32,32,2)
output = similar(input)
sparsity_pattern = jacobian_sparsity(brusselator_2d_loop,output,input,p,0.0)
jac_sparsity = Float64.(sparse(sparsity_pattern))
```

```
Explored path: SparsityDetection.Path(Bool[], 1)
2048×2048 SparseArrays.SparseMatrixCSC{Float64, Int64} with 12288 stored en
tries:
```

Let's double check what our sparsity pattern looks like:

```
using Plots
spy(jac_sparsity,markersize=1,colorbar=false,color=:deep)
```



That's neat, and would be tedius to build by hand! Now we just pass it to the `ODEFunction` like as before:

```
f = ODEFunction(brusselator_2d_loop;jac_prototype=jac_sparsity)
```

```
(::SciMLBase.ODEFunction{true, typeof(Main.##WeaveSandBox#5739.brusselator_
2d_loop), LinearAlgebra.UniformScaling{Bool}, Nothing, Nothing, Nothing, No
thing, Nothing, SparseArrays.SparseMatrixCSC{Float64, Int64}, SparseArrays.
SparseMatrixCSC{Float64, Int64}, Nothing, Nothing, Nothing, Nothing, Nothin
g, typeof(SciMLBase.DEFAULT_OBSERVED), Nothing}) (generic function with 7 m
ethods)
```

Build the `ODEProblem`:

```
function init_brusselator_2d(xyd)
  N = length(xyd)
  u = zeros(N, N, 2)
  for I in CartesianIndices((N, N))
    x = xyd[I[1]]
    y = xyd[I[2]]
    u[I,1] = 22*(y*(1-y))^(3/2)
    u[I,2] = 27*(x*(1-x))^(3/2)
  end
  u
end
u0 = init_brusselator_2d(xyd_brusselator)
prob_ode_brusselator_2d = ODEProblem(brusselator_2d_loop,
                                     u0,(0.,11.5),p)

prob_ode_brusselator_2d_sparse = ODEProblem(f,
                                     u0,(0.,11.5),p)
```

```
ODEProblem with uType Array{Float64, 3} and tType Float64. In-place: true
timespan: (0.0, 11.5)
```

```
u0: 32×32×2 Array{Float64, 3}:
[:, :, 1] =
 0.0  0.121344  0.326197  0.568534  …  0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534  …  0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 ⋮                                  ⋱                       ⋮
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534  …  0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534  …  0.568534  0.326197  0.121344  0.0
 0.0  0.121344  0.326197  0.568534     0.568534  0.326197  0.121344  0.0

[:, :, 2] =
 0.0       0.0       0.0       0.0       …  0.0       0.0       0.0
 0.148923  0.148923  0.148923  0.148923     0.148923  0.148923  0.148923
 0.400332  0.400332  0.400332  0.400332     0.400332  0.400332  0.400332
 0.697746  0.697746  0.697746  0.697746     0.697746  0.697746  0.697746
 1.01722   1.01722   1.01722   1.01722      1.01722   1.01722   1.01722
 1.34336   1.34336   1.34336   1.34336   …  1.34336   1.34336   1.34336
 1.66501   1.66501   1.66501   1.66501      1.66501   1.66501   1.66501
 1.97352   1.97352   1.97352   1.97352      1.97352   1.97352   1.97352
 2.26207   2.26207   2.26207   2.26207      2.26207   2.26207   2.26207
 2.52509   2.52509   2.52509   2.52509      2.52509   2.52509   2.52509
 ⋮                                       ⋱            ⋮
 2.26207   2.26207   2.26207   2.26207      2.26207   2.26207   2.26207
 1.97352   1.97352   1.97352   1.97352      1.97352   1.97352   1.97352
 1.66501   1.66501   1.66501   1.66501   …  1.66501   1.66501   1.66501
 1.34336   1.34336   1.34336   1.34336      1.34336   1.34336   1.34336
 1.01722   1.01722   1.01722   1.01722      1.01722   1.01722   1.01722
 0.697746  0.697746  0.697746  0.697746     0.697746  0.697746  0.697746
 0.400332  0.400332  0.400332  0.400332     0.400332  0.400332  0.400332
 0.148923  0.148923  0.148923  0.148923  …  0.148923  0.148923  0.148923
 0.0       0.0       0.0       0.0          0.0       0.0       0.0
```

Now let's see how the version with sparsity compares to the version without:

```julia
@btime solve(prob_ode_brusselator_2d,save_everystep=false)
@btime solve(prob_ode_brusselator_2d_sparse,save_everystep=false)
```

```
4.026 s (3332 allocations: 65.33 MiB)
  868.275 ms (40171 allocations: 276.18 MiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 … 0.1213443281371586 0.0; 0.0 0.12134432813715876
```

```
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [3.8715710568026327 3.871544263496401 ... 3.871660597887853 3.87161004723348
5; 3.8716190219250093 3.871588988900678 ... 3.871720060854604 3.8716628901712
69; ... ; 3.8714871831703883 3.8714656453085934 ... 3.8715582925263354 3.871518
307861871; 3.871526626222637 3.8715026809065862 ... 3.871606147539579 3.87156
13475793575]

[1.5025267482810192 1.5025277497723653 ... 1.5025234812450186 1.5025253112096
277; 1.5025247560530617 1.502525831533873 ... 1.502521238116099 1.50252321042
43587; ... ; 1.5025302969061514 1.5025311733894735 ... 1.5025274521973424 1.502
5290429883629; 1.502528617794096 1.5025295523513722 ... 1.502525577078082 1.5
025272788129502]
```

0.2.5 Declaring Color Vectors for Fast Construction

If you cannot directly define a Jacobian function, you can use the `colorvec` to speed up the Jacobian construction. What the `colorvec` does is allows for calculating multiple columns of a Jacobian simultaniously by using the sparsity pattern. An explanation of matrix coloring can be found in the MIT 18.337 Lecture Notes.

To perform general matrix coloring, we can use SparseDiffTools.jl. For example, for the Brusselator equation:

```
using SparseDiffTools
colorvec = matrix_colors(jac_sparsity)
@show maximum(colorvec)
```

```
maximum(colorvec) = 12
12
```

This means that we can now calculate the Jacobian in 12 function calls. This is a nice reduction from 2048 using only automated tooling! To now make use of this inside of the ODE solver, you simply need to declare the colorvec:

```
f = ODEFunction(brusselator_2d_loop;jac_prototype=jac_sparsity,
                                    colorvec=colorvec)
prob_ode_brusselator_2d_sparse = ODEProblem(f,
                                     init_brusselator_2d(xyd_brusselator),
                                     (0.,11.5),p)
@btime solve(prob_ode_brusselator_2d_sparse,save_everystep=false)
```

```
865.774 ms (7390 allocations: 272.21 MiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432813715876
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
```

```
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [3.8715710568026327 3.871544263496401 ... 3.871660597887853 3.87161004723348
5; 3.8716190219250093 3.871588988900678 ... 3.871720060854604 3.8716628901712
69; ... ; 3.8714871831703883 3.8714656453085934 ... 3.8715582925263354 3.871518
307861871; 3.871526626222637 3.8715026809065862 ... 3.871606147539579 3.87156
13475793575]

[1.5025267482810192 1.5025277497723653 ... 1.5025234812450186 1.5025253112096
277; 1.5025247560530617 1.502525831533873 ... 1.502521238116099 1.50252321042
43587; ... ; 1.5025302969061514 1.5025311733894735 ... 1.5025274521973424 1.502
5290429883629; 1.502528617794096 1.5025295523513722 ... 1.502525577078082 1.5
025272788129502]
```

Notice the massive speed enhancement!

0.3 Defining Linear Solver Routines and Jacobian-Free Newton-Krylov

A completely different way to optimize the linear solvers for large sparse matrices is to use a Krylov subpsace method. This requires choosing a linear solver for changing to a Krylov method. Optionally, one can use a Jacobian-free operator to reduce the memory requirements.

0.3.1 Declaring a Jacobian-Free Newton-Krylov Implementation

To swap the linear solver out, we use the `linsolve` command and choose the GMRES linear solver.

```
@btime
solve(prob_ode_brusselator_2d,TRBDF2(linsolve=LinSolveGMRES()),save_everystep=false)
@btime
solve(prob_ode_brusselator_2d_sparse,TRBDF2(linsolve=LinSolveGMRES()),save_everystep=false)
```

```
53.459 s (1440760 allocations: 148.08 MiB)
  3.459 s (487052 allocations: 19.49 MiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432813715876
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [1.4496467189952293 1.4496188458395953 ... 1.449739625696683 1.4496857634543
15; 1.4496924266563709 1.4496627513504483 ... 1.4497954395000603 1.4497385989
710139; ... ; 1.4495499647945365 1.4495293829622544 ... 1.4496249819457812 1.44
95821445340045; 1.4495986625150836 1.4495728479552343 ... 1.449681691859659 1
.4496342425553288]
```

```
[4.555791737526942 4.555792854178718 ... 4.555785210283977 4.555788777078638;
 4.555787105443905 4.555788047943169 ... 4.555781401248265 4.5557847763611905
; ... ; 4.5558058525318765 4.555807862015024 ... 4.5558015022696345 4.555804088
530327; 4.555797778755576 4.555798553618496 ... 4.5557925979760325 4.55579571
470298]
```

For more information on linear solver choices, see the linear solver documentation.

On this problem, handling the sparsity correctly seemed to give much more of a speedup than going to a Krylov approach, but that can be dependent on the problem (and whether a good preconditioner is found).

We can also enhance this by using a Jacobian-Free implementation of `f'(x)*v`. To define the Jacobian-Free operator, we can use DiffEqOperators.jl to generate an operator `JacVecOperator` such that `Jv*v` performs `f'(x)*v` without building the Jacobian matrix.

```
using DiffEqOperators
Jv = JacVecOperator(brusselator_2d_loop,u0,p,0.0)
```

```
DiffEqOperators.JacVecOperator{Float64, typeof(Main.##WeaveSandBox#5739.bru
sselator_2d_loop), Array{ForwardDiff.Dual{DiffEqOperators.JacVecTag, Float6
4, 1}, 3}, Array{ForwardDiff.Dual{DiffEqOperators.JacVecTag, Float64, 1}, 3
}, Array{Float64, 3}, NTuple{4, Float64}, Float64, Bool}(Main.##WeaveSandBo
x#5739.brusselator_2d_loop, ForwardDiff.Dual{DiffEqOperators.JacVecTag, Flo
at64, 1}[Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacV
ecTag}(0.12134432813715876,0.12134432813715876) ... Dual{DiffEqOperators.JacV
ecTag}(0.1213443281371586,0.1213443281371586) Dual{DiffEqOperators.JacVecTa
g}(0.0,0.0); Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.
JacVecTag}(0.12134432813715876,0.12134432813715876) ... Dual{DiffEqOperators.
JacVecTag}(0.1213443281371586,0.1213443281371586) Dual{DiffEqOperators.JacV
ecTag}(0.0,0.0); ... ; Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOp
erators.JacVecTag}(0.12134432813715876,0.12134432813715876) ... Dual{DiffEqOp
erators.JacVecTag}(0.1213443281371586,0.1213443281371586) Dual{DiffEqOperat
ors.JacVecTag}(0.0,0.0); Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{Diff
EqOperators.JacVecTag}(0.12134432813715876,0.12134432813715876) ... Dual{Diff
EqOperators.JacVecTag}(0.1213443281371586,0.1213443281371586) Dual{DiffEqOp
erators.JacVecTag}(0.0,0.0)]

ForwardDiff.Dual{DiffEqOperators.JacVecTag, Float64, 1}[Dual{DiffEqOperator
s.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0) ... Dual{DiffE
qOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0); Du
al{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) Dual
{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) ... Dual
{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) Dual{D
iffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755); ... ; Dua
l{DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) Dual{
DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) ... Dual{
DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) Dual{Di
ffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738); Dual{Dif
fEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0) ...
 Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0
.0,0.0)], ForwardDiff.Dual{DiffEqOperators.JacVecTag, Float64, 1}[Dual{Diff
EqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.121344328
13715876,0.12134432813715876) ... Dual{DiffEqOperators.JacVecTag}(0.121344328
1371586,0.1213443281371586) Dual{DiffEqOperators.JacVecTag}(0.0,0.0); Dual{
DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.12134
432813715876,0.12134432813715876) ... Dual{DiffEqOperators.JacVecTag}(0.12134
43281371586,0.1213443281371586) Dual{DiffEqOperators.JacVecTag}(0.0,0.0); ...
 ; Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}
(0.12134432813715876,0.12134432813715876) ... Dual{DiffEqOperators.JacVecTag}
```

```
(0.1213443281371586,0.1213443281371586) Dual{DiffEqOperators.JacVecTag}(0.0
,0.0); Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVec
Tag}(0.12134432813715876,0.12134432813715876) ... Dual{DiffEqOperators.JacVec
Tag}(0.1213443281371586,0.1213443281371586) Dual{DiffEqOperators.JacVecTag}
(0.0,0.0)]

ForwardDiff.Dual{DiffEqOperators.JacVecTag, Float64, 1}[Dual{DiffEqOperator
s.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0) ... Dual{DiffE
qOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0); Du
al{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) Dual
{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) ... Dual
{DiffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755) Dual{D
iffEqOperators.JacVecTag}(0.14892258453196755,0.14892258453196755); ... ; Dua
l{DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) Dual{
DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) ... Dual{
DiffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738) Dual{Di
ffEqOperators.JacVecTag}(0.14892258453196738,0.14892258453196738); Dual{Dif
fEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0.0,0.0) ...
 Dual{DiffEqOperators.JacVecTag}(0.0,0.0) Dual{DiffEqOperators.JacVecTag}(0
.0,0.0)], [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432
813715876 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443
281371586 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0], (3.4, 1.0, 10.0
, 0.03225806451612903), 0.0, true, false, true)
```

and then we can use this by making it our `jac_prototype`:

```
f = ODEFunction(brusselator_2d_loop;jac_prototype=Jv)
prob_ode_brusselator_2d_jacfree = ODEProblem(f,u0,(0.,11.5),p)
@btime
solve(prob_ode_brusselator_2d_jacfree,TRBDF2(linsolve=LinSolveGMRES()),save_everystep=false)
```

```
2.149 s (942433 allocations: 1.05 GiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432813715876
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [1.328086637873347 1.328059197713509 ... 1.3281748347697229 1.32812606585897
29; 1.328130848643362 1.3280992632574358 ... 1.3282306297437354 1.32817542936
69061; ... ; 1.3280077761453217 1.3279862091179504 ... 1.3280777261839196 1.328
0384277888153; 1.3280454775601096 1.3280204995302134 ... 1.328123305013178 1.
3280799053895886]

[4.6985106146296785 4.698511510656312 ... 4.698506092751332 4.698508731423445
; 4.698506492475369 4.698507558534346 ... 4.698502182960649 4.698504769487088
; ... ; 4.698517566578984 4.698518821016911 ... 4.698513034821239 4.69851561364
0851; 4.698514120257303 4.698515413421847 ... 4.698509865089128 4.69851240717
4246]
```

0.3.2 Adding a Preconditioner

The linear solver documentation shows how you can add a preconditioner to the GMRES. For example, you can use packages like AlgebraicMultigrid.jl to add an algebraic multigrid (AMG) or IncompleteLU.jl for an incomplete LU-factorization (iLU).

```
using AlgebraicMultigrid
pc = aspreconditioner(ruge_stuben(jac_sparsity))
@btime
solve(prob_ode_brusselator_2d_jacfree,TRBDF2(linsolve=LinSolveGMRES(Pl=pc)),save_everystep=false)
```

```
52.889 ms (2126 allocations: 4.62 MiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432813715876
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [10517.228691133823 10903.17821877683 ... 9234.374974925357 13421.8684240780
77; 14610.689352333644 8520.29499343432 ... 9234.400192154684 13421.868424078
073; ... ; 13421.868424078082 9234.400192154602 ... 9234.40019215468 13421.8684
24078077; 13421.86842407808 9234.37497492528 ... 9234.374974925358 13421.8684
24078077]

[16505.210468729245 16435.39296876962 ... 16462.923992780543 16458.1794295503
68; 11307.018407220907 11343.187214827063 ... 11331.237752550098 11326.518406
549272; ... ; 11326.518406549352 11331.23775255019 ... 11331.237752550187 11326
.518406549356; 16458.179429550346 16462.923992780536 ... 16462.923992780536 1
6458.179429550346]
```

0.4 Using Structured Matrix Types

If your sparsity pattern follows a specific structure, for example a banded matrix, then you can declare `jac_prototype` to be of that structure and then additional optimizations will come for free. Note that in this case, it is not necessary to provide a `colorvec` since the color vector will be analytically derived from the structure of the matrix.

The matrices which are allowed are those which satisfy the ArrayInterface.jl interface for automatically-colorable matrices. These include:

- Bidiagonal
- Tridiagonal
- SymTridiagonal
- BandedMatrix (BandedMatrices.jl)
- BlockBandedMatrix (BlockBandedMatrices.jl)

Matrices which do not satisfy this interface can still be used, but the matrix coloring will not be automatic, and an appropriate linear solver may need to be given (otherwise it will default to attempting an LU-decomposition).

0.5 Sundials-Specific Handling

While much of the setup makes the transition to using Sundials automatic, there are some differences between the pure Julia implementations and the Sundials implementations which must be taken note of. These are all detailed in the Sundials solver documentation, but here we will highlight the main details which one should make note of.

Defining a sparse matrix and a Jacobian for Sundials works just like any other package. The core difference is in the choice of the linear solver. With Sundials, the linear solver choice is done with a Symbol in the `linear_solver` from a preset list. Particular choices of note are `:Band` for a banded matrix and `:GMRES` for using GMRES. If you are using Sundials, `:GMRES` will not require defining the JacVecOperator, and instead will always make use of a Jacobian-Free Newton Krylov (with numerical differentiation). Thus on this problem we could do:

```
using Sundials
# Sparse Version
@btime solve(prob_ode_brusselator_2d_sparse,CVODE_BDF(),save_everystep=false)
# GMRES Version: Doesn't require any extra stuff!
@btime
solve(prob_ode_brusselator_2d,CVODE_BDF(linear_solver=:GMRES),save_everystep=false)
```

```
15.577 s (51406 allocations: 3.40 MiB)
  297.422 ms (54356 allocations: 3.24 MiB)
retcode: Success
Interpolation: 1st order linear
t: 2-element Vector{Float64}:
  0.0
 11.5
u: 2-element Vector{Array{Float64, 3}}:
 [0.0 0.12134432813715876 ... 0.1213443281371586 0.0; 0.0 0.12134432813715876
 ... 0.1213443281371586 0.0; ... ; 0.0 0.12134432813715876 ... 0.1213443281371586
 0.0; 0.0 0.12134432813715876 ... 0.1213443281371586 0.0]

[0.0 0.0 ... 0.0 0.0; 0.14892258453196755 0.14892258453196755 ... 0.14892258453
196755 0.14892258453196755; ... ; 0.14892258453196738 0.14892258453196738 ... 0
.14892258453196738 0.14892258453196738; 0.0 0.0 ... 0.0 0.0]
 [0.73954982624037 0.7395263388402707 ... 0.7396351353060872 0.73958594992628
53; 0.7396041658726187 0.7395794572427602 ... 0.739695044334048 0.73964269792
96304; ... ; 0.7394464539934248 0.7394236442964711 ... 0.739525711935924 0.7394
816883777938; 0.7394971311298292 0.7394736007090181 ... 0.7395776392281666 0.
7395310908993405]

[5.2303939324919755 5.230393890057542 ... 5.2303959984740604 5.23039480844585
2; 5.230381468535975 5.230378860934974 ... 5.23039032191521 5.230385777894235
; ... ; 5.230417475089114 5.230421848938212 ... 5.2304069866992995 5.2304127960
05578; 5.230406186817664 5.230408247767747 ... 5.230402329506947 5.2304044894
92586]
```

Details for setting up a preconditioner with Sundials can be found at the Sundials solver page.

0.6 Handling Mass Matrices

Instead of just defining an ODE as $u' = f(u,p,t)$, it can be common to express the differential equation in the form with a mass matrix:

$$Mu' = f(u,p,t)$$

where M is known as the mass matrix. Let's solve the Robertson equation. At the top we wrote this equation as:

$$\begin{align} dy_1 &= -0.04y\_1 + 10^4 y_2 y_3 \tag{5}\\ dy_2 &= 0.04y_1 - 10^4 y_2 y_3 - 3 * 10^7 y_2^2 \tag{6}\\ dy_3 &= 3 * 10^7 y_3^2 \tag{7}\\ & \tag{8}\end{align}$$

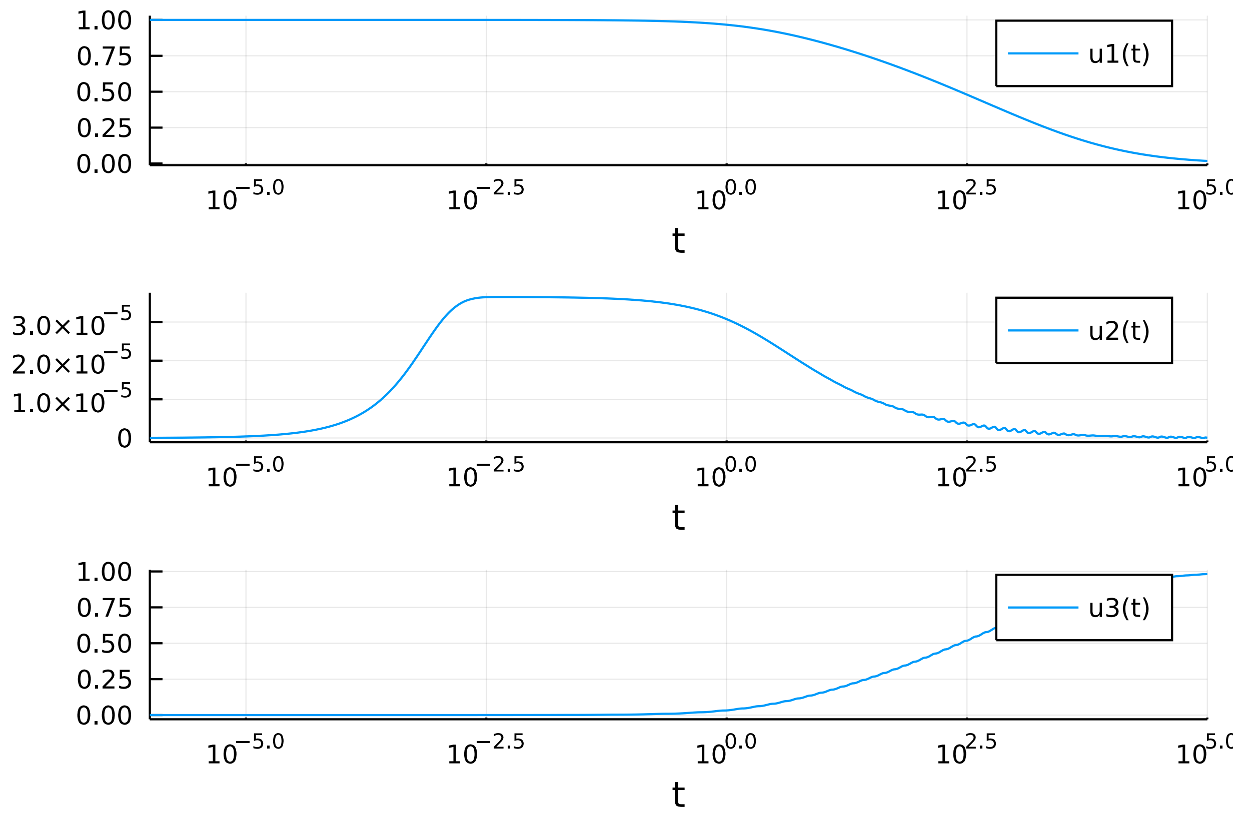
But we can instead write this with a conservation relation:

$$\begin{align} dy_1 &= -0.04y\_1 + 10^4 y_2 y_3 \tag{9}\\ dy_2 &= 0.04y_1 - 10^4 y_2 y_3 - 3 * 10^7 y_2^2 \tag{10}\\ 1 &= y_1 + y_2 + y_3 \tag{11}\\ & \tag{12}\end{align}$$

In this form, we can write this as a mass matrix ODE where M is singular (this is another form of a differential-algebraic equation (DAE)). Here, the last row of `M` is just zero. We can implement this form as:

```
using DifferentialEquations
function rober(du,u,p,t)
  y_1,y_2,y_3 = u
  k_1,k_2,k_3 = p
  du[1] = -k_1*y_1+k_3*y_2*y_3
  du[2] =  k_1*y_1-k_2*y_2^2-k_3*y_2*y_3
  du[3] =  y_1 + y_2 + y_3 - 1
  nothing
end
M = [1. 0  0
     0  1. 0
     0  0  0]
f = ODEFunction(rober,mass_matrix=M)
prob_mm = ODEProblem(f,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
sol = solve(prob_mm,Rodas5())

plot(sol, xscale=:log10, tspan=(1e-6, 1e5), layout=(3,1))
```



Note that if your mass matrix is singular, i.e. your system is a DAE, then you need to make sure you choose a solver that is compatible with DAEs

0.7 Appendix

These tutorials are a part of the SciMLTutorials.jl repository, found at: https://github.com/SciML/SciML For more information on high-performance scientific machine learning, check out the SciML Open Source Software Organization https://sciml.ai.

To locally run this tutorial, do the following commands:

```
using SciMLTutorials
SciMLTutorials.weave_file("tutorials/advanced","02-advanced_ODE_solving.jmd")
```

Computer Information:

```
Julia Version 1.6.2
Commit 1b93d53fc4 (2021-07-14 15:36 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: AMD EPYC 7502 32-Core Processor
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-11.0.1 (ORCJIT, znver2)
Environment:
  JULIA_DEPOT_PATH = /root/.cache/julia-buildkite-plugin/depots/a6029d3a-f78b-41ea-bc9
```

```
JULIA_NUM_THREADS = 16
```

Package Information:

```
    Status `/var/lib/buildkite-agent/builds/8-amdci4-julia-csail-mit-edu/julialang/sc
[2169fc97] AlgebraicMultigrid v0.4.0
[6e4b80f9] BenchmarkTools v1.0.0
[052768ef] CUDA v2.6.3
[2b5f629d] DiffEqBase v6.62.2
[9fdde737] DiffEqOperators v4.26.0
[0c46a032] DifferentialEquations v6.17.1
[587475ba] Flux v0.12.1
[961ee093] ModelingToolkit v5.17.3
[2774e3e8] NLsolve v4.5.1
[315f7962] NeuralPDE v3.10.1
[1dea7af3] OrdinaryDiffEq v5.56.0
[91a5bcdd] Plots v1.15.2
[0bca4576] SciMLBase v1.13.4
[30cb0354] SciMLTutorials v0.9.0
[47a9eef4] SparseDiffTools v1.13.2
[684fba80] SparsityDetection v0.3.4
[789caeaf] StochasticDiffEq v6.34.1
[c3572dad] Sundials v4.4.3
[37e2e46d] LinearAlgebra
[2f01184e] SparseArrays
```

And the full manifest:

```
    Status `/var/lib/buildkite-agent/builds/8-amdci4-julia-csail-mit-edu/julialang/sc
[c3fe647b] AbstractAlgebra v0.16.0
[621f4979] AbstractFFTs v1.0.1
[1520ce14] AbstractTrees v0.3.4
[79e6a3ab] Adapt v3.3.0
[2169fc97] AlgebraicMultigrid v0.4.0
[ec485272] ArnoldiMethod v0.1.0
[4fba245c] ArrayInterface v3.1.15
[4c555306] ArrayLayouts v0.7.0
[13072b0f] AxisAlgorithms v1.0.0
[ab4f0b2a] BFloat16s v0.1.0
[aae01518] BandedMatrices v0.16.9
[6e4b80f9] BenchmarkTools v1.0.0
[8e7c35d0] BlockArrays v0.15.3
[ffab5731] BlockBandedMatrices v0.10.6
[764a87c0] BoundaryValueDiffEq v2.7.1
[fa961155] CEnum v0.4.1
[00ebfdb7] CSTParser v2.5.0
[052768ef] CUDA v2.6.3
```

```
[7057c7e9] Cassette v0.3.6
[082447d4] ChainRules v0.7.65
[d360d2e6] ChainRulesCore v0.9.44
[b630d9fa] CheapThreads v0.2.5
[944b1d66] CodecZlib v0.7.0
[35d6a980] ColorSchemes v3.12.1
[3da002f7] ColorTypes v0.11.0
[5ae59095] Colors v0.12.8
[861a8166] Combinatorics v1.0.2
[a80b9123] CommonMark v0.8.1
[38540f10] CommonSolve v0.2.0
[bbf7d656] CommonSubexpressions v0.3.0
[34da2185] Compat v3.30.0
[aa819f21] CompatHelper v1.18.6
[8f4d0f93] Conda v1.5.2
[88cd18e8] ConsoleProgressMonitor v0.1.2
[187b0558] ConstructionBase v1.2.1
[d38c429a] Contour v0.5.7
[a8cc5b0e] Crayons v4.0.4
[8a292aeb] Cuba v2.2.0
[667455a9] Cubature v1.5.1
[9a962f9c] DataAPI v1.6.0
[82cc6244] DataInterpolations v3.3.1
[864edb3b] DataStructures v0.18.9
[e2d170a0] DataValueInterfaces v1.0.0
[bcd4f6db] DelayDiffEq v5.31.0
[2b5f629d] DiffEqBase v6.62.2
[459566f4] DiffEqCallbacks v2.16.1
[5a0ffddc] DiffEqFinancial v2.4.0
[aae7a2af] DiffEqFlux v1.37.0
[c894b116] DiffEqJump v6.14.2
[77a26b50] DiffEqNoiseProcess v5.7.3
[9fdde737] DiffEqOperators v4.26.0
[055956cb] DiffEqPhysics v3.9.0
[41bf760c] DiffEqSensitivity v6.45.0
[163ba53b] DiffResults v1.0.3
[b552c78f] DiffRules v1.0.2
[0c46a032] DifferentialEquations v6.17.1
[c619ae07] DimensionalPlotRecipes v1.2.0
[b4f34e82] Distances v0.10.3
[31c24e10] Distributions v0.24.18
[ced4e74d] DistributionsAD v0.6.26
[ffbed154] DocStringExtensions v0.8.4
[e30172f5] Documenter v0.26.3
[d4d017d3] ExponentialUtilities v1.8.4
[e2ba6199] ExprTools v0.1.3
[8f5d6c58] EzXML v1.1.0
[c87230d0] FFMPEG v0.4.0
[7a1cc6ca] FFTW v1.4.1
```

```
[7034ab61] FastBroadcast v0.1.8
[9aa1b823] FastClosures v0.3.2
[1a297f60] FillArrays v0.11.7
[6a86dc24] FiniteDiff v2.8.0
[53c48c17] FixedPointNumbers v0.8.4
[587475ba] Flux v0.12.1
[59287772] Formatting v0.4.2
[f6369f11] ForwardDiff v0.10.18
[069b7b12] FunctionWrappers v1.1.2
[d9f16b24] Functors v0.2.1
[0c68f7d7] GPUArrays v6.4.1
[61eb1bfa] GPUCompiler v0.10.0
[28b8d3ca] GR v0.57.4
[a75be94c] GalacticOptim v1.2.0
[5c1252a2] GeometryBasics v0.3.12
[bc5e4493] GitHub v5.4.0
[af5da776] GlobalSensitivity v1.0.0
[42e2da0e] Grisu v1.0.2
[19dc6840] HCubature v1.5.0
[cd3eb016] HTTP v0.9.9
[eafb193a] Highlights v0.4.5
[0e44f5e4] Hwloc v2.0.0
[7073ff75] IJulia v1.23.2
[b5f81e59] IOCapture v0.1.1
[7869d1d1] IRTools v0.4.2
[615f187c] IfElse v0.1.0
[d25df0c9] Inflate v0.1.2
[83e8ac13] IniFile v0.5.0
[a98d9a8b] Interpolations v0.13.2
[c8e1da08] IterTools v1.3.0
[42fd0dbc] IterativeSolvers v0.9.1
[82899510] IteratorInterfaceExtensions v1.0.0
[692b3bcd] JLLWrappers v1.3.0
[682c06a0] JSON v0.21.1
[98e50ef6] JuliaFormatter v0.13.7
[e5e0dc1b] Juno v0.8.4
[5ab0869b] KernelDensity v0.6.3
[929cbde3] LLVM v3.7.1
[b964fa9f] LaTeXStrings v1.2.1
[2ee39098] LabelledArrays v1.6.1
[23fbe1c1] Latexify v0.15.5
[a5e1c1ea] LatinHypercubeSampling v1.8.0
[73f95e8e] LatticeRules v0.0.1
[5078a376] LazyArrays v0.21.4
[d7e5e226] LazyBandedMatrices v0.5.7
[1d6d02ad] LeftChildRightSiblingTrees v0.1.2
[093fc24a] LightGraphs v1.3.5
[d3d80556] LineSearches v7.1.1
[2ab3a3ac] LogExpFunctions v0.2.4
```

```
[e6f89c97] LoggingExtras v0.4.6
[bdcacae8] LoopVectorization v0.12.23
[1914dd2f] MacroTools v0.5.6
[a3b82374] MatrixFactorizations v0.8.3
[739be429] MbedTLS v1.0.3
[442fdcdd] Measures v0.3.1
[e89f7d12] Media v0.5.0
[c03570c3] Memoize v0.4.4
[e1d29d7a] Missings v1.0.0
[78c3b35d] Mocking v0.7.1
[961ee093] ModelingToolkit v5.17.3
[4886b29c] MonteCarloIntegration v0.0.2
[46d2c3a1] MuladdMacro v0.2.2
[f9640e96] MultiScaleArrays v1.8.1
[ffc61752] Mustache v1.0.10
[d41bc354] NLSolversBase v7.8.0
[2774e3e8] NLsolve v4.5.1
[872c559c] NNlib v0.7.19
[77ba4419] NaNMath v0.3.5
[315f7962] NeuralPDE v3.10.1
[8913a72c] NonlinearSolve v0.3.8
[6fe1bfb0] OffsetArrays v1.9.0
[429524aa] Optim v1.3.0
[bac558e1] OrderedCollections v1.4.1
[1dea7af3] OrdinaryDiffEq v5.56.0
[90014a1f] PDMats v0.11.0
[65888b18] ParameterizedFunctions v5.10.0
[d96e819e] Parameters v0.12.2
[69de0a69] Parsers v1.1.0
[ccf2f8ad] PlotThemes v2.0.1
[995b91a9] PlotUtils v1.0.10
[91a5bcdd] Plots v1.15.2
[e409e4f3] PoissonRandom v0.4.0
[f517fe37] Polyester v0.3.1
[85a6dd25] PositiveFactorizations v0.2.4
[21216c6a] Preferences v1.2.2
[33c8b6b6] ProgressLogging v0.1.4
[92933f4c] ProgressMeter v1.6.2
[1fd47b50] QuadGK v2.4.1
[67601950] Quadrature v1.8.1
[8a4e6c94] QuasiMonteCarlo v0.2.2
[74087812] Random123 v1.3.1
[fb686558] RandomExtensions v0.4.3
[e6cf234a] RandomNumbers v1.4.0
[c84ed2f1] Ratios v0.4.0
[3cdcf5f2] RecipesBase v1.1.1
[01d81517] RecipesPipeline v0.3.2
[731186ca] RecursiveArrayTools v2.11.4
[f2c3362d] RecursiveFactorization v0.1.12
```

```
[189a3867] Reexport v1.0.0
[ae029012] Requires v1.1.3
[ae5879a3] ResettableStacks v1.1.0
[37e2e3b7] ReverseDiff v1.9.0
[79098fc4] Rmath v0.7.0
[7e49a35a] RuntimeGeneratedFunctions v0.5.2
[476501e8] SLEEFPirates v0.6.20
[1bc83da4] SafeTestsets v0.0.1
[0bca4576] SciMLBase v1.13.4
[30cb0354] SciMLTutorials v0.9.0
[6c6a2e73] Scratch v1.0.3
[efcf1570] Setfield v0.7.0
[992d4aef] Showoff v1.0.3
[699a6c99] SimpleTraits v0.9.3
[ed01d8cd] Sobol v1.5.0
[2133526b] SodiumSeal v0.1.1
[b85f4697] SoftGlobalScope v1.1.0
[a2af1166] SortingAlgorithms v1.0.0
[47a9eef4] SparseDiffTools v1.13.2
[684fba80] SparsityDetection v0.3.4
[276daf66] SpecialFunctions v1.4.1
[860ef19b] StableRNGs v1.0.0
[aedffcd0] Static v0.2.4
[90137ffa] StaticArrays v1.2.0
[82ae8749] StatsAPI v1.0.0
[2913bbd2] StatsBase v0.33.8
[4c63d2b9] StatsFuns v0.9.8
[9672c7b4] SteadyStateDiffEq v1.6.2
[789caeaf] StochasticDiffEq v6.34.1
[7792a7ef] StrideArraysCore v0.1.11
[09ab397b] StructArrays v0.5.1
[c3572dad] Sundials v4.4.3
[d1185830] SymbolicUtils v0.11.2
[0c5d862f] Symbolics v0.1.25
[3783bdb8] TableTraits v1.0.1
[bd369af6] Tables v1.4.2
[5d786b92] TerminalLoggers v0.1.3
[8290d209] ThreadingUtilities v0.4.4
[f269a46b] TimeZones v1.5.5
[a759f4b9] TimerOutputs v0.5.9
[0796e94c] Tokenize v0.5.16
[9f7883ad] Tracker v0.2.16
[3bb67fe8] TranscodingStreams v0.9.5
[592b5752] Trapz v2.0.2
[a2a6695c] TreeViews v0.3.0
[5c2747f8] URIs v1.3.0
[3a884ed6] UnPack v1.0.2
[1986cc42] Unitful v1.7.0
[3d5dd08c] VectorizationBase v0.20.11
```

```
[81def892] VersionParsing v1.2.0
[19fa3120] VertexSafeGraphs v0.1.2
[44d3d7a6] Weave v0.10.8
[efce3f68] WoodburyMatrices v0.5.3
[ddb6d928] YAML v0.4.6
[c2297ded] ZMQ v1.2.1
[a5390f91] ZipFile v0.9.3
[e88e6eb3] Zygote v0.6.11
[700de1a5] ZygoteRules v0.2.1
[6e34b625] Bzip2_jll v1.0.6+5
[83423d85] Cairo_jll v1.16.0+6
[3bed1096] Cuba_jll v4.2.1+0
[7bc98958] Cubature_jll v1.0.4+0
[5ae413db] EarCut_jll v2.1.5+1
[2e619515] Expat_jll v2.2.10+0
[b22a6f82] FFMPEG_jll v4.3.1+4
[f5851436] FFTW_jll v3.3.9+7
[a3f928ae] Fontconfig_jll v2.13.1+14
[d7e528f0] FreeType2_jll v2.10.1+5
[559328eb] FriBidi_jll v1.0.5+6
[0656b61e] GLFW_jll v3.3.4+0
[d2c73de3] GR_jll v0.57.2+0
[78b55507] Gettext_jll v0.21.0+0
[7746bdde] Glib_jll v2.68.1+0
[e33a78d0] Hwloc_jll v2.4.1+0
[1d5cc7b8] IntelOpenMP_jll v2018.0.3+2
[aacddb02] JpegTurbo_jll v2.0.1+3
[c1c5ebd0] LAME_jll v3.100.0+3
[dd4b983a] LZO_jll v2.10.1+0
[dd192d2f] LibVPX_jll v1.9.0+1
[e9f186c6] Libffi_jll v3.2.2+0
[d4300ac3] Libgcrypt_jll v1.8.7+0
[7e76a0d4] Libglvnd_jll v1.3.0+3
[7add5ba3] Libgpg_error_jll v1.42.0+0
[94ce4f54] Libiconv_jll v1.16.1+0
[4b2f31a3] Libmount_jll v2.35.0+0
[89763e89] Libtiff_jll v4.1.0+2
[38a345b3] Libuuid_jll v2.36.0+0
[856f044c] MKL_jll v2021.1.1+1
[e7412a2a] Ogg_jll v1.3.4+2
[458c3c95] OpenSSL_jll v1.1.1+6
[efe28fd5] OpenSpecFun_jll v0.5.4+0
[91d4177d] Opus_jll v1.3.1+3
[2f80f16e] PCRE_jll v8.44.0+0
[30392449] Pixman_jll v0.40.1+0
[ea2cea3b] Qt5Base_jll v5.15.2+0
[f50d1b31] Rmath_jll v0.3.0+0
[fb77eaff] Sundials_jll v5.2.0+1
[a2964d1f] Wayland_jll v1.17.0+4
```

```
[2381bf8a] Wayland_protocols_jll v1.18.0+4
[02c8fc9c] XML2_jll v2.9.12+0
[aed1982a] XSLT_jll v1.1.34+0
[4f6342f7] Xorg_libX11_jll v1.6.9+4
[0c0b7dd1] Xorg_libXau_jll v1.0.9+4
[935fb764] Xorg_libXcursor_jll v1.2.0+4
[a3789734] Xorg_libXdmcp_jll v1.1.3+4
[1082639a] Xorg_libXext_jll v1.3.4+4
[d091e8ba] Xorg_libXfixes_jll v5.0.3+4
[a51aa0fd] Xorg_libXi_jll v1.7.10+4
[d1454406] Xorg_libXinerama_jll v1.1.4+4
[ec84b674] Xorg_libXrandr_jll v1.5.2+4
[ea2f1a96] Xorg_libXrender_jll v0.9.10+4
[14d82f49] Xorg_libpthread_stubs_jll v0.1.0+3
[c7cfdc94] Xorg_libxcb_jll v1.13.0+3
[cc61e674] Xorg_libxkbfile_jll v1.1.0+4
[12413925] Xorg_xcb_util_image_jll v0.4.0+1
[2def613f] Xorg_xcb_util_jll v0.4.0+1
[975044d2] Xorg_xcb_util_keysyms_jll v0.4.0+1
[0d47668e] Xorg_xcb_util_renderutil_jll v0.3.9+1
[c22f9ab0] Xorg_xcb_util_wm_jll v0.4.1+1
[35661453] Xorg_xkbcomp_jll v1.4.2+4
[33bec58e] Xorg_xkeyboard_config_jll v2.27.0+4
[c5fb5394] Xorg_xtrans_jll v1.4.0+3
[8f1865be] ZeroMQ_jll v4.3.2+6
[3161d3a3] Zstd_jll v1.5.0+0
[0ac62f75] libass_jll v0.14.0+4
[f638f0a6] libfdk_aac_jll v0.1.6+4
[b53b4c65] libpng_jll v1.6.38+0
[a9144af2] libsodium_jll v1.0.20+0
[f27f6e37] libvorbis_jll v1.3.6+6
[1270edf5] x264_jll v2020.7.14+2
[dfaa095f] x265_jll v3.0.0+3
[d8fb68d0] xkbcommon_jll v0.9.1+5
[0dad84c5] ArgTools
[56f22d72] Artifacts
[2a0f44e3] Base64
[ade2ca70] Dates
[8bb1440f] DelimitedFiles
[8ba89e20] Distributed
[f43a241f] Downloads
[7b1f6079] FileWatching
[9fa8497b] Future
[b77e0a4c] InteractiveUtils
[4af54fe1] LazyArtifacts
[b27032c2] LibCURL
[76f85450] LibGit2
[8f399da3] Libdl
[37e2e46d] LinearAlgebra
```

```
[56ddb016] Logging
[d6f4376e] Markdown
[a63ad114] Mmap
[ca575930] NetworkOptions
[44cfe95a] Pkg
[de0858da] Printf
[9abbd945] Profile
[3fa0cd96] REPL
[9a3f8284] Random
[ea8e919c] SHA
[9e88b42a] Serialization
[1a1011a3] SharedArrays
[6462fe0b] Sockets
[2f01184e] SparseArrays
[10745b16] Statistics
[4607b0f0] SuiteSparse
[fa267f1f] TOML
[a4e569a6] Tar
[8dfed614] Test
[cf7118a7] UUIDs
[4ec0a83e] Unicode
[e66e0078] CompilerSupportLibraries_jll
[deac9b47] LibCURL_jll
[29816b5a] LibSSH2_jll
[c8ffd9c3] MbedTLS_jll
[14a3606d] MozillaCACerts_jll
[4536629a] OpenBLAS_jll
[bea87d4a] SuiteSparse_jll
[83775a58] Zlib_jll
[8e850ede] nghttp2_jll
[3f19e933] p7zip_jll
```