Summaries

July 14, 2015

1 Summaries of timing tests

We compare several optimizers, in both R and Julia, fitting a selection of linear mixed models. In **R** the optimizers are called by `lmer` from the lme4 package (version 1.1-8). In **Julia** the `lmm` function from the MixedModels package calls the optimizers.

There are differences in the model formulations in `lme4` and in `MixedModels`. The numerical representation of the model in `lme4` and the method of evaluating the objective, described in this paper, is the same for all models. In `MixedModels` there are specialized representations for some model forms, such as models with a single grouping factor for the random effects. Some of the specialized representations allow for evaluation of the gradient of the objects, which can enhance convergence (but, interestingly, sometimes can impede convergence).

1.1 Methodology

To provide consistency we have copied all the data sets used in the timings to the `Timings` package itself. We have done all timings on the same computer. This computer has a relatively recent Intel processor and we used the Intel Math Kernel Library (MKL) with Julia. We attempted to use Revolution R Open (RRO) as the R implementation as it can be configured with MKL. However, we ran into version problems with this so we used the standard Ubuntu version of R linked against OpenBLAS, which is also multi-threaded.

Variables were renamed in the pattern: - **Y** the response - **A**, **B**, ... categorical covariates - **G**, **H**, **I**, ... grouping factors for random effects - **U**, **V**, ... (skipping **Y**) continuous covariates

The timing results are saved in JSON (JavaScript Object Notation) files in the directory accessible as

```
system.file("JSON",package="Timings")
```

within **R**. The directory name will end with `./Timings/inst/JSON/` in the package source directory, for example the result of cloning the github repository. There is one `.json` file for each data set. Each such file contains results on timings of one or more models.

The `Timings` package for **R** provides a `retime` function that takes the name of one of these JSON files and, optionally, the name of a file with the updated timings. Similarly there are some source files for Julia retimings.

```
In [1]: include("../julia/retime.jl")
        retime("../JSON/Alfalfa.json","/tmp/Alfalfa.json");
        retime("../JSON/Alfalfa.json","/tmp/Alfalfa.json");
```

```
dsname => "Alfalfa"


        `parse` has no method matching parse(::Int64)
    while loading In[1], in expression starting on line 2



         in retime at /home/bates/git/Timings/inst/julia/retime.jl:17
```

The timing was repeated so that compilation time is not included in the results. This repetition is only needed once per session.

A careful examination of these results shows that the main differences in the Julia timings (the R timings are merely reported, not evaluated) are that the `LN_BOBYQA` and `LD_MMA` optimizers are much faster in the second run. This is because much of the code needs to be compiled the first time that a derivative-free optimizer and a derivative-based optimizer are used.

The names of the optimizers used with `lmm` are those from the NLopt package for **Julia**. Names that begin with `LD_` are gradient-based methods. Names that begin with `LN_` are derivative-free methods. There is one other derivative-free method, `LN_PRAXIS`, available in the `NLopt` package but, for some reason, it can hang on very simple problems like this. Frequently we omit it.

The optimizers used with `lmer` include the `Nelder_Mead` optimizer built into the `lme4` package, the `bobyqa` optimizer from the minqa package, the derivative-free optimizers from the nloptr package and several optimizers from the optimx pacakge.

The `optimx:bobyqa` optimizer is just a wrapper around `bobyqa` (bounded optimization by quadratic approximation) from the `minqa` package and should provide results similar to those from the `bobyqa` optimizer. For some reason the number of function evaluations is not reported for the version in `optimx`.

The optimizers from `nloptr` (i.e. those whose names begin with `NLOPT_LN_`) use the same underlying code as do the similarly named optimizers in the `NLopt` package for **Julia**. The number of iterations to convergence should be similar for the same underlying code, although not nessarily exactly the same because the evaluation of the objective in **R** and in **Julia** may produce slightly different answers. Also the convergence criteria in the **Julia** version are more strict than those in the **R** version

Also shown are the value of the criterion (negative twice the log-likelihood, lower is better) achieved, the elapsed time and the number of function and gradient evaluations. The `nopt` value is the number of parameters in the optimization problem. `mtype` is the model type in the **Julia** code. There are special methods for solving the penalized least squares (PLS) problem, and for evaluating the objective and its gradient when there is only one grouping factor for the random effects. The model type is called `PLSOne`.

The `Alfalfa` example is a particularly easy one and all of the optimizerws converge to an objective value close to -10.81023 in less than 0.6 seconds.

1.1.1 Tabulating results

For the `Alfalfa` data there is not much of a burden in refitting the model with all the **Julia** optimizers just to get the table shown above. But other examples can take an hour or more to converge and we don't really need to refit them every time. The `tabulate.jl` file contains a function `optdir` to create a `DataFrame` from the results of all the model fits.

In [2]:
```
include("../julia/tabulate.jl")
res = optdir("../JSON");
res[1:30,[1,2,3,6,7,8,9]]
```

Out[2]:
```
30x7 DataFrame
| Row | opt        | dsname           | n      | np | excess  | time    |
|-----|------------|------------------|--------|----|---------|---------|
| 1   | "LD_CCSAQ" | "Alfalfa"        | 72     | 1  | 0.0     | 0.0017  |
| 2   | "LD_CCSAQ" | "AvgDailyGain"   | 32     | 1  | 0.0     | 0.0014  |
| 3   | "LD_CCSAQ" | "AvgDailyGain"   | 32     | 1  | 0.0     | 0.0014  |
| 4   | "LD_CCSAQ" | "BIB"            | 24     | 1  | 0.0     | 0.0013  |
| 5   | "LD_CCSAQ" | "Bond"           | 21     | 1  | 0.0     | 0.0009  |
| 6   | "LD_CCSAQ" | "bs10"           | 1104   | 20 | 0.0     | 1.0958  |
| 7   | "LD_CCSAQ" | "bs10"           | 1104   | 8  | 39.9948 | 0.0375  |
| 8   | "LD_CCSAQ" | "cake"           | 270    | 1  | 0.0     | 0.0033  |
| 9   | "LD_CCSAQ" | "Cultivation"    | 24     | 1  | 0.0     | 0.0009  |
| 10  | "LD_CCSAQ" | "Demand"         | 77     | 2  | 3.21928 | 0.0055  |
| 11  | "LD_CCSAQ" | "dialectNL"      | 225866 | 6  | 0.0     | 6.9896  |
⋮
```

```
| 19  | "LD_CCSAQ" | "gb12"          | 512     | 8  | 103.176 | 0.0218  |
| 20  | "LD_CCSAQ" | "HR"            | 120     | 3  | 0.0     | 0.0089  |
| 21  | "LD_CCSAQ" | "Hsb82"         | 7185    | 1  | 192.73  | 0.0102  |
| 22  | "LD_CCSAQ" | "IncBlk"        | 24      | 1  | 0.55726 | 0.001   |
| 23  | "LD_CCSAQ" | "kb07"          | 1790    | 72 | 8.20739 | 17.4698 |
| 24  | "LD_CCSAQ" | "Mississippi"   | 37      | 1  | 0.93471 | 0.0006  |
| 25  | "LD_CCSAQ" | "mm0"           | 69588   | 6  | 0.0     | 4.8286  |
| 26  | "LD_CCSAQ" | "Oxboys"        | 234     | 3  | 136.788 | 0.0169  |
| 27  | "LD_CCSAQ" | "PBIB"          | 60      | 1  | 0.0     | 0.0014  |
| 28  | "LD_CCSAQ" | "Penicillin"    | 144     | 2  | 0.0     | 0.0131  |
| 29  | "LD_CCSAQ" | "Semiconductor" | 48      | 1  | 0.0     | 0.0012  |
| 30  | "LD_CCSAQ" | "SIMS"          | 3691    | 3  | 3.60856 | 0.134   |

| Row | reltime |
|-----|---------|
| 1   | 1.1342  |
| 2   | 0.8207  |
| 3   | 0.9283  |
| 4   | 0.7897  |
| 5   | 1.0265  |
| 6   | 4.4375  |
| 7   | 0.555   |
| 8   | 1.3714  |
| 9   | 0.9925  |
| 10  | 0.8079  |
| 11  | 3.9932  |
⋮
| 19  | 0.5604  |
| 20  | 1.2196  |
| 21  | 0.4919  |
| 22  | 0.6573  |
| 23  | 4.1242  |
| 24  | 0.6716  |
| 25  | 4.4016  |
| 26  | 0.7092  |
| 27  | 0.9986  |
| 28  | 4.9414  |
| 29  | 1.0461  |
| 30  | 0.9858  |
```

The `time` column is the time in seconds to converge. The `reltime` column is the time relative to the `LN_BOBYQA` optimizer in the `MixedModels` package for **Julia**.

```
In [3]: res[res[:opt] .== "NLOPT_LN_BOBYQA",[1,2,3,6,7,8,9]]
```

```
Out[3]: 49x7 DataFrame
| Row | opt               | dsname          | n       | np | excess | time   |
|-----|-------------------|-----------------|---------|----|--------|--------|
| 1   | "NLOPT_LN_BOBYQA" | "Alfalfa"       | 72      | 1  | 0.0    | 0.042  |
| 2   | "NLOPT_LN_BOBYQA" | "Animal"        | 20      | 2  | 0.0    | 0.023  |
| 3   | "NLOPT_LN_BOBYQA" | "Assay"         | 60      | 2  | 1.0e-5 | 0.032  |
| 4   | "NLOPT_LN_BOBYQA" | "AvgDailyGain"  | 32      | 1  | 0.0    | 0.02   |
| 5   | "NLOPT_LN_BOBYQA" | "AvgDailyGain"  | 32      | 1  | 0.0    | 0.02   |
| 6   | "NLOPT_LN_BOBYQA" | "BIB"           | 24      | 1  | 0.0    | 0.02   |
| 7   | "NLOPT_LN_BOBYQA" | "Bond"          | 21      | 1  | 0.0    | 0.02   |
```

```
| 8   | "NLOPT_LN_BOBYQA" | "bs10"          | 1104   | 20 | 1.0e-5 | 4.661  |
| 9   | "NLOPT_LN_BOBYQA" | "bs10"          | 1104   | 8  | 0.0    | 1.057  |
| 10  | "NLOPT_LN_BOBYQA" | "cake"          | 270    | 1  | 0.0    | 0.053  |
| 11  | "NLOPT_LN_BOBYQA" | "Chem97"        | 31022  | 2  | 0.0    | 0.632  |
⋮
| 38  | "NLOPT_LN_BOBYQA" | "PBIB"          | 60     | 1  | 0.0    | 0.018  |
| 39  | "NLOPT_LN_BOBYQA" | "Penicillin"    | 144    | 2  | 0.0    | 0.023  |
| 40  | "NLOPT_LN_BOBYQA" | "Poems"         | 275996 | 3  | 0.0    | 21.309 |
| 41  | "NLOPT_LN_BOBYQA" | "ScotsSec"      | 3435   | 2  | 0.0    | 0.076  |
| 42  | "NLOPT_LN_BOBYQA" | "Semi2"         | 72     | 3  | 0.0    | 0.03   |
| 43  | "NLOPT_LN_BOBYQA" | "Semiconductor" | 48     | 1  | 0.0    | 0.019  |
| 44  | "NLOPT_LN_BOBYQA" | "SIMS"          | 3691   | 3  | 0.0    | 0.15   |
| 45  | "NLOPT_LN_BOBYQA" | "sleepstudy"    | 180    | 3  | 0.0    | 0.037  |
| 46  | "NLOPT_LN_BOBYQA" | "sleepstudy"    | 180    | 2  | 0.0    | 0.024  |
| 47  | "NLOPT_LN_BOBYQA" | "TeachingII"    | 96     | 1  | 0.0    | 0.021  |
| 48  | "NLOPT_LN_BOBYQA" | "Weights"       | 399    | 3  | 1.0e-5 | 0.039  |
| 49  | "NLOPT_LN_BOBYQA" | "WWheat"        | 60     | 3  | 0.0    | 0.025  |

| Row | reltime |
|-----|---------|
| 1   | 27.5171 |
| 2   | 13.9236 |
| 3   | 10.8009 |
| 4   | 11.7005 |
| 5   | 13.6969 |
| 6   | 12.608  |
| 7   | 23.2699 |
| 8   | 18.8753 |
| 9   | 15.6477 |
| 10  | 22.0827 |
| 11  | 3.942   |
⋮
| 38  | 13.1358 |
| 39  | 8.6481  |
| 40  | 3.7438  |
| 41  | 5.0422  |
| 42  | 9.2732  |
| 43  | 15.9582 |
| 44  | 1.1034  |
| 45  | 5.3039  |
| 46  | 8.6177  |
| 47  | 15.3763 |
| 48  | 1.133   |
| 49  | 2.3154  |
```

1.2 Proportion converged

The most important question regarding the optimizers is whether or not they have converged to the global optimum. We cannot test this directly. Instead we use a "crowd-sourced" criterion based on the minimum objective achieved by any of the algorithms. The difference between the objective achieved by a particular algorithm and this minimum is called the `excess`. In the summaries `excess` is rounded to 5 digits after the decimal so the minimum non-zero `excess` is 10^{-5}.

```
In [4]: res[res[:opt] .== "LN_BOBYQA",[:opt,:dsname,:excess]]
```

Out[4]: 49x3 DataFrame

```
| Row | opt         | dsname           | excess |
|-----|-------------|------------------|--------|
| 1   | "LN_BOBYQA" | "Alfalfa"        | 0.0    |
| 2   | "LN_BOBYQA" | "Animal"         | 0.0    |
| 3   | "LN_BOBYQA" | "Assay"          | 0.0    |
| 4   | "LN_BOBYQA" | "AvgDailyGain"   | 0.0    |
| 5   | "LN_BOBYQA" | "AvgDailyGain"   | 0.0    |
| 6   | "LN_BOBYQA" | "BIB"            | 0.0    |
| 7   | "LN_BOBYQA" | "Bond"           | 0.0    |
| 8   | "LN_BOBYQA" | "bs10"           | 1.0e-5 |
| 9   | "LN_BOBYQA" | "bs10"           | 0.0    |
| 10  | "LN_BOBYQA" | "cake"           | 0.0    |
| 11  | "LN_BOBYQA" | "Chem97"         | 0.0    |
⋮
| 38  | "LN_BOBYQA" | "PBIB"           | 0.0    |
| 39  | "LN_BOBYQA" | "Penicillin"     | 0.0    |
| 40  | "LN_BOBYQA" | "Poems"          | 0.0    |
| 41  | "LN_BOBYQA" | "ScotsSec"       | 0.0    |
| 42  | "LN_BOBYQA" | "Semi2"          | 0.0    |
| 43  | "LN_BOBYQA" | "Semiconductor"  | 0.0    |
| 44  | "LN_BOBYQA" | "SIMS"           | 0.0    |
| 45  | "LN_BOBYQA" | "sleepstudy"     | 0.0    |
| 46  | "LN_BOBYQA" | "sleepstudy"     | 0.0    |
| 47  | "LN_BOBYQA" | "TeachingII"     | 0.0    |
| 48  | "LN_BOBYQA" | "Weights"        | 0.0    |
| 49  | "LN_BOBYQA" | "WWheat"         | 0.0    |
```

If we wish to declare "converged" or "not converged" according to the excess objective value we must establish a threshold. An absolute threshold seems reasonable because the objective, negative twice the log-likelihood, is on a scale where differences in this objective are compared to a χ^2 random variable. Thus an excess of 10^{-9} or even 10^{-5} is negligible.

For each optimizer we can examine which of the data set/model combinations resulted in an excess greater than a threshold.

In [5]:

```
by(res,:opt) do df
    DataFrame(attempted=size(df,1),failed=countnz(df[:excess] .> 0.02))
end
```

Out[5]: 26x3 DataFrame

```
| Row | opt                           | attempted | failed |
|-----|-------------------------------|-----------|--------|
| 1   | "LD_CCSAQ"                    | 35        | 11     |
| 2   | "LD_LBFGS"                    | 35        | 11     |
| 3   | "LD_MMA"                      | 36        | 5      |
| 4   | "LD_SLSQP"                    | 35        | 4      |
| 5   | "LD_TNEWTON"                  | 34        | 8      |
| 6   | "LD_TNEWTON_PRECOND"          | 34        | 8      |
| 7   | "LD_TNEWTON_PRECOND_RESTART"  | 33        | 12     |
| 8   | "LD_TNEWTON_RESTART"          | 34        | 11     |
| 9   | "LD_VAR1"                     | 35        | 10     |
| 10  | "LD_VAR2"                     | 35        | 10     |
| 11  | "LN_BOBYQA"                   | 49        | 0      |
⋮
```

```
| 15  | "LN_SBPLX"                    | 49        | 2         |
| 16  | "NLOPT_LN_BOBYQA"             | 49        | 0         |
| 17  | "NLOPT_LN_COBYLA"             | 48        | 2         |
| 18  | "NLOPT_LN_NELDERMEAD"         | 47        | 6         |
| 19  | "NLOPT_LN_PRAXIS"             | 18        | 4         |
| 20  | "NLOPT_LN_SBPLX"              | 48        | 2         |
| 21  | "Nelder_Mead"                 | 49        | 8         |
| 22  | "bobyqa"                      | 49        | 2         |
| 23  | "optimx:L-BFGS-B"             | 49        | 0         |
| 24  | "optimx:bobyqa"               | 49        | 2         |
| 25  | "optimx:nlminb"               | 49        | 0         |
| 26  | "optimx:spg"                  | 49        | 4         |
```

At this threshold the most reliable algorithm in Julia is `LN_BOBYQA`. In R the most reliable algorithms are `NLOPT_LN_BOBYQA`, `optimx:L-BFGS-B` and `optimx:nlminb`. It is interesting that nlminb is reliable as I felt that it wasn't converging well when it was the default optimizer in `lmer`.

Interestingly, the derviative-based algorithms in `NLopt` were not as reliable as the derivative-free algorithms. The most likely explanation is that I don't have the gradient coded properly.

The Nelder-Mead simplex algorithm did not perform well, failing on 8 out of 48 cases. For many of these the value at which convergence was declared was far from the optimum.

In [6]:
```
noncvg =
by(res,:opt) do df
    df[df[:excess] .> 0.005,[:dsname,:excess,:time,:reltime,:np,:n]]
end;
dfselect(df::AbstractDataFrame,col::Symbol,val) = df[df[col] .== val, :]
dfselect(noncvg,:opt,"Nelder_Mead")
```

Out[6]:
```
8x7 DataFrame
| Row | opt           | dsname        | excess  | time    | reltime | np |
|-----|---------------|---------------|---------|---------|---------|----|
| 1   | "Nelder_Mead" | "bs10"        | 71.3859 | 145.368 | 588.684 | 20 |
| 2   | "Nelder_Mead" | "d3"          | 317.59  | 454.519 | 4.2502  | 9  |
| 3   | "Nelder_Mead" | "dialectNL"   | 181.632 | 54.541  | 31.1594 | 6  |
| 4   | "Nelder_Mead" | "gb12"        | 78.7119 | 38.206  | 189.605 | 20 |
| 5   | "Nelder_Mead" | "kb07"        | 398.732 | 2825.46 | 667.015 | 72 |
| 6   | "Nelder_Mead" | "kb07"        | 403.478 | 269.436 | 383.087 | 16 |
| 7   | "Nelder_Mead" | "Mississippi" | 0.04272 | 0.018   | 20.3989 | 1  |
| 8   | "Nelder_Mead" | "mm0"         | 181.632 | 76.87   | 70.0716 | 6  |

| Row | n      |
|-----|--------|
| 1   | 1104   |
| 2   | 130418 |
| 3   | 225866 |
| 4   | 512    |
| 5   | 1790   |
| 6   | 1790   |
| 7   | 37     |
| 8   | 69588  |
```

In [7]:
```
dfselect(noncvg,:opt,"NLOPT_LN_NELDERMEAD")
```

Out[7]:
```
6x7 DataFrame
| Row | opt                   | dsname  | excess  | time    | reltime | np |
```

```
|-----|------------------------|---------|---------|---------|---------|----|
| 1   | "NLOPT_LN_NELDERMEAD" | "Assay" | 0.05942 | 0.042   | 14.1762 | 2  |
| 2   | "NLOPT_LN_NELDERMEAD" | "bs10"  | 0.97096 | 88.286  | 357.524 | 20 |
| 3   | "NLOPT_LN_NELDERMEAD" | "d3"    | 100.191 | 627.931 | 5.8718  | 9  |
| 4   | "NLOPT_LN_NELDERMEAD" | "gb12"  | 0.83883 | 52.411  | 260.1   | 20 |
| 5   | "NLOPT_LN_NELDERMEAD" | "kb07"  | 88.7686 | 2839.07 | 670.229 | 72 |
| 6   | "NLOPT_LN_NELDERMEAD" | "kb07"  | 3.49868 | 198.845 | 282.72  | 16 |

| Row | n      |
|-----|--------|
| 1   | 60     |
| 2   | 1104   |
| 3   | 130418 |
| 4   | 512    |
| 5   | 1790   |
| 6   | 1790   |
```

The `Nelder_Mead` algorithm, either in the native form in `lmer` or in the `NLopt` implementation performed poorly on those cases with many parameters to optimize. It was both unreliable and slow, taking over 45 minutes to reach a spurious optimum on the "maximal" model (in the sense of Barr et al., 2012) for the `kb07` data from Kronmueller and Barr (2007). This is not terribly surprising given that the model is horribly overparameterized, but still it shows that this algorithm is not a good choice in these cases.

We note in passing that all the models involving fitting 20 or more parameters are "maximal" models in the sense of Barr et al., 2012. Such models can present difficult optimization problems because they are severely overparameterized and inevitably converge on the boundary of the allowable parameter space. Whether or not it is sensible to compare results on such extreme cases is not clear.

The `SBPLX` (subplex) algorithm, which is an enhancement of `Nelder_Mead`, does better in these cases but is still rather slow.

```
In [8]: dfselect(noncvg,:opt,"NLOPT_LN_SBPLX")
```

```
Out[8]: 2x7 DataFrame
        | Row | opt              | dsname | excess  | time    | reltime | np | n    |
        |-----|------------------|--------|---------|---------|---------|----|------|
        | 1   | "NLOPT_LN_SBPLX" | "gb12" | 0.82219 | 3.813   | 18.9228 | 20 | 512  |
        | 2   | "NLOPT_LN_SBPLX" | "kb07" | 4.96546 | 564.688 | 133.308 | 72 | 1790 |
```

By comparison, the `LN_BOBYQA` algorithm converges quite rapidly on the `kb07` models.

```
In [9]: bobyqa = res[convert(Array,res[:opt] .== "LN_BOBYQA") &
            convert(Array,res[:dsname] .== "kb07"),
            [:dsname,:excess,:objective,:time,:np]]
```

```
Out[9]: 2x5 DataFrame
        | Row | dsname | excess  | objective | time   | np |
        |-----|--------|---------|-----------|--------|----|
        | 1   | "kb07" | 0.01695 | 28586.3   | 4.236  | 72 |
        | 2   | "kb07" | 0.0     | 28670.9   | 0.7033 | 16 |
```

1.3 Relative speed

We plot the time to convergence, relative to `LN_BOBYQA` and on a logarithmic scale, for each algorithm.

```
In [10]: using Gadfly
         set_default_plot_size(16cm,12cm)
         res[:cvg] = compact(pool([e > 0.02 ? "N" : "Y" for e in res[:excess]]))
```

```
plot(res,x="reltime",y="opt",color="cvg",
    Geom.point,Scale.y_discrete,Scale.x_log2,
    Guide.ylabel(nothing),
    Guide.xlabel("Time to convergence relative to LN_BOBYQA"))
```

Out[10]:

max size=0.90.9Summaries$_{f}iles/Summaries_{1}8_{0}.pdf$

Many of the cases where `LN_BOBYQA` is slower than other algorithms are simple problems that converge in less than 1/5 of a second for most algorithms.

We will declare a data set to be non-simple if at least one of the models fit to the data took more than 0.2 seconds to convergence with `LN_BOBYQA`.

In [12]:
```
ln_bobyqa = dfselect(res,:opt,"LN_BOBYQA");
nonsimple = ln_bobyqa[ln_bobyqa[:time] .> 0.2,[:dsname,:time,:n,:np,:models]]
```

Out[12]:
```
10x5 DataFrame
| Row | dsname      | time   | n      | np |
|-----|-------------|--------|--------|----|
| 1   | "bs10"      | 0.2469 | 1104   | 20 |
| 2   | "d3"        | 106.94 | 130418 | 9  |
| 3   | "dialectNL" | 1.7504 | 225866 | 6  |
| 4   | "gb12"      | 0.2015 | 512    | 20 |
| 5   | "InstEval"  | 2.3539 | 73421  | 2  |
| 6   | "InstEval"  | 4.3593 | 73421  | 3  |
| 7   | "kb07"      | 4.236  | 1790   | 72 |
| 8   | "kb07"      | 0.7033 | 1790   | 16 |
| 9   | "mm0"       | 1.097  | 69588  | 6  |
| 10  | "Poems"     | 5.6918 | 275996 | 3  |

| Row | models
|-----|--------------------------------------------------------------------------------------------
| 1   | "29"
| 2   | "Y ~ U + (U | G)  + (U | H) + (U | I)"
| 3   | "Y ~ U + V +W +X + Z+A + T +(1 | G) +(0 + V + W+ X | G) +(1 + Z + A | H) +(1 | I)"
| 4   | "Y ~ 1+S+T+U+V+W+X+Z + (1+S+U+W|G) + (1+S+T+V|H)"
| 5   | "Y ~ 1 + I*A + (1|G) + (1|H)"
| 6   | "Y ~ 1 + A + (1|G) + (1|H) + (1|I)"
| 7   | "Y ~ 1+S+T+U+V+W+X+Z + (1+S+T+U+V+W+X+Z|G) + (1+S+T+U+V+W+X+Z|H)"
| 8   | "Y ~ 1+S+T+U+V+W+X+Z + (1|G)+(0+S|G)+(0+T|G)+(0+U|G)+(0+V|G)+(0+W|G)+(0+X|G)+(0+Z|G) +
| 9   | "Y~A*U+(1+U|G)+(1+U|H)"
| 10  | "Y ~ 1 + U + V + (1|G) + (1|H) + (1|I)"
```

Notice that these are cases with a large number of observations (`n`) or a large number of parameters in the optimization problem (`np`) or both.

By comparison, the cases where other algorithms are faster than `LN_BOBYQA` are, for the most part, models and data sets with few observations and few parameters to optimize. In these circumstances almost all the optimizers are fast.

In [13]:
```
by(res,:opt) do df
    cvgfast = (df[:reltime] .< 1) & convert(Array,df[:cvg] .== "Y")
    if any(cvgfast)
        print(df[cvgfast,[:opt,:dsname,:time,:np,:n]])
        println()
    end
end;
```

8x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_CCSAQ"	"AvgDailyGain"	0.0014	1	32
2	"LD_CCSAQ"	"AvgDailyGain"	0.0014	1	32
3	"LD_CCSAQ"	"BIB"	0.0013	1	24
4	"LD_CCSAQ"	"Cultivation"	0.0009	1	24
5	"LD_CCSAQ"	"Dyestuff2"	0.0005	1	30
6	"LD_CCSAQ"	"Gasoline"	0.0011	1	32
7	"LD_CCSAQ"	"PBIB"	0.0014	1	60
8	"LD_CCSAQ"	"TeachingII"	0.0013	1	96

2x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_LBFGS"	"gb12"	0.1871	20	512
2	"LD_LBFGS"	"HR"	0.004	3	120

12x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_MMA"	"AvgDailyGain"	0.0015	1	32
2	"LD_MMA"	"BIB"	0.0012	1	24
3	"LD_MMA"	"Bond"	0.0008	1	21
4	"LD_MMA"	"Dyestuff2"	0.0005	1	30
5	"LD_MMA"	"Dyestuff"	0.0007	1	30
6	"LD_MMA"	"ergoStool"	0.0009	1	36
7	"LD_MMA"	"Gasoline"	0.0012	1	32
8	"LD_MMA"	"Oxboys"	0.0092	3	234
9	"LD_MMA"	"PBIB"	0.0013	1	60
10	"LD_MMA"	"TeachingII"	0.0012	1	96
11	"LD_MMA"	"Weights"	0.027	3	399
12	"LD_MMA"	"WWheat"	0.0084	3	60

20x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_SLSQP"	"Alfalfa"	0.0014	1	72
2	"LD_SLSQP"	"AvgDailyGain"	0.0016	1	32
3	"LD_SLSQP"	"BIB"	0.0011	1	24
4	"LD_SLSQP"	"Bond"	0.0007	1	21
5	"LD_SLSQP"	"Cultivation"	0.0008	1	24
6	"LD_SLSQP"	"Demand"	0.0054	2	77
7	"LD_SLSQP"	"Dyestuff2"	0.0007	1	30
8	"LD_SLSQP"	"Dyestuff"	0.0007	1	30
9	"LD_SLSQP"	"ergoStool"	0.001	1	36
10	"LD_SLSQP"	"Gasoline"	0.0011	1	32
11	"LD_SLSQP"	"HR"	0.0056	3	120
12	"LD_SLSQP"	"IncBlk"	0.0013	1	24
13	"LD_SLSQP"	"Oxboys"	0.0072	3	234
14	"LD_SLSQP"	"PBIB"	0.0009	1	60
15	"LD_SLSQP"	"Semiconductor"	0.001	1	48
16	"LD_SLSQP"	"SIMS"	0.0742	3	3691
17	"LD_SLSQP"	"sleepstudy"	0.0052	3	180
18	"LD_SLSQP"	"TeachingII"	0.0011	1	96
19	"LD_SLSQP"	"Weights"	0.0234	3	399
20	"LD_SLSQP"	"WWheat"	0.0079	3	60

3x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_TNEWTON"	"HR"	0.0059	3	120
2	"LD_TNEWTON"	"Oxboys"	0.0087	3	234
3	"LD_TNEWTON"	"Weights"	0.0213	3	399

3x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_TNEWTON_PRECOND"	"HR"	0.0048	3	120
2	"LD_TNEWTON_PRECOND"	"Oxboys"	0.0092	3	234
3	"LD_TNEWTON_PRECOND"	"Weights"	0.0152	3	399

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_TNEWTON_PRECOND_RESTART"	"HR"	0.005	3	120

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_TNEWTON_RESTART"	"HR"	0.0052	3	120

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_VAR1"	"HR"	0.0044	3	120

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"LD_VAR2"	"HR"	0.0044	3	120

2x5 DataFrame

Row	opt	dsname	time	np	n
1	"LN_COBYLA"	"BIB"	0.0014	1	24
2	"LN_COBYLA"	"Hsb82"	0.018	1	7185

3x5 DataFrame

Row	opt	dsname	time	np	n
1	"LN_NELDERMEAD"	"BIB"	0.0015	1	24
2	"LN_NELDERMEAD"	"Dyestuff2"	0.0005	1	30
3	"LN_NELDERMEAD"	"Oxboys"	0.0192	3	234

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"LN_PRAXIS"	"Poems"	5.3089	3	275996

4x5 DataFrame

Row	opt	dsname	time	np	n
1	"LN_SBPLX"	"d3"	75.8816	9	130418
2	"LN_SBPLX"	"Demand"	0.0049	2	77
3	"LN_SBPLX"	"dialectNL"	1.4855	6	225866
4	"LN_SBPLX"	"Dyestuff2"	0.0006	1	30

1x5 DataFrame

Row	opt	dsname	time	np	n
1	"NLOPT_LN_COBYLA"	"SIMS"	0.092	3	3691