Solving sudoku's with R and C++

Bob Jansen

SatrDay September 1, 2018

What is a sudoku

The goal of a sudoku puzzle is to fill a 9×9 grid of cells with numbers. At the start the grid will be partially filled and the solver must fill all the empty cells according to a set of simple rules:

- Cells may only be filled by the numbers 1 to 9.
- A number may appear only once in each row.
- A number may appear only once in each column.
- The grid is divided in a 3×3 grid of 3×3 cells each. In each smaller grid a number may only appear once.

Solving Sudoko's in R and C++

In this presentation two approaches are compared to solve sudoku's:

- R
- Hybrid of R and C++

Representing a sudoku

A sudoku will be represented by a space separated numbers in 9 rows and 9 columns. For example (taken from Wikipedia):

```
sudokuTxt <- "
1 0 0  0 0 0  0 0 6
0 0 6  0 2 0  7 0 0
7 8 9  4 5 0  1 0 3

0 0 0  8 0 7  0 0 4
0 0 0  0 3 0  0 0 0
0 9 0  0 0 4  2 0 1

3 1 2  9 7 0  0 4 0
0 4 0  0 1 2  0 7 8
9 0 8  0 0 0  0 0 0"
```

0 signifies an empty cell.

Loading the sudoku

Reading in the sudoku and storing it in a matrix using R is simple:

```
sudoku <- as.matrix(
  read.table(text = sudokuTxt,
             col.names = letters[1:9]))
```

General solution strategy

A simple idea to algorithm to solve sudoku's is to pick an empty cell fill it with a valid choice and then repeat this process.

- `findChoices()` is a function that finds possible values for empty cells given the current grid of numbers.
- A contradiction is found when for a partial solution a cell has *no* valid choices. In this case, this partial solution can never be made a correct solution by filling in further cells.

In R code

```r
solve <- function(partialSolution, choicesFUN) {
  # Eliminate impossible values, give some suggestions
  # and flag contradictions.
  c(partialSolution, suggestions, contradiction) %<-%
    eliminate(partialSolution, choicesFUN)
  # If dead end FALSE to trace back, if finished TRUE.
  if (contradiction) return(list(FALSE, NULL))
  if (all(partialSolution %in% 1:9))
    return(list(TRUE, partialSolution))
  # Branching, exit when the solution is found.
  for (suggestion in suggestions) {
    c(result, solution) %<-% solve(suggestion,
                                   choicesFUN)
    if (result) return(list(result, solution))
  }
  list(FALSE, NULL)
}
```

R Implementation - `eliminate()`

The function `eliminate()` checks for every empty cell which values are possible, returns a contradiction when no possible values for a cell are found and keeps track of the cell with the least possibilities which can be used as a pivot. If only one value is possible it is filled in directly.

R Implementation - eliminate()

```r
eliminate <- function(grid, choicesFUN) {
  suggestions <- 0:9
  for (i in 1:nrow(grid)) { for (j in 1:ncol(grid)) {
    if (grid[i, j] == 0L) {
      choices <- choicesFUN(grid, i, j)
      if (length(choices) == 0L) {
        return(list(NULL, NULL, TRUE))
      } else if (length(choices) == 1L) {
        grid[i, j] <- choices
        return(list(grid, list(grid), FALSE))
      } else
        suggestions <- updateSuggestions(
          choices, grid, i, j, suggestions)
    }
  }}
  list(grid, suggestions, FALSE)
}
```

R Implementation - Helpers

```r
# Find all the choices allowed by the rules.
findChoices <- function(grid, i, j) {
  1:9 %>% setdiff(grid[i, ]) %>%
    setdiff(grid[ , j]) %>%
    setdiff(grid[i - (i - 1) %% 3L + 0:2,
                 j - (j - 1) %% 3L + 0:2])
}
# Create a list of grids with suggested next moves.
updateSuggestions <- function(choices, grid, i, j,
                              lastBest) {
  if (length(choices) < length(lastBest))
    lapply(choices, function(choice) {
      grid[i, j] <- choice; grid
    })
  else
    lastBest
}
```

Solving sudoku's

With this code solving a sudoko is simple:

```
solution <- solve(sudoku, findChoices)
if (!solution[[1]]) { cat('Solution not found\n')
} else { print(as.data.frame(solution[[2]])) }
```

```
##   a b c d e f g h i
## 1 1 2 3 7 8 9 4 5 6
## 2 4 5 6 1 2 3 7 8 9
## 3 7 8 9 4 5 6 1 2 3
## 4 2 3 1 8 9 7 5 6 4
## 5 5 6 4 2 3 1 8 9 7
## 6 8 9 7 5 6 4 2 3 1
## 7 3 1 2 9 7 8 6 4 5
## 8 6 4 5 3 1 2 9 7 8
## 9 9 7 8 6 4 5 3 1 2
```

or as a picture

1	2	3	7	8	9	4	5	6
4	5	6	1	2	3	7	8	9
7	8	9	4	5	6	1	2	3
2	3	1	8	9	7	5	6	4
5	6	4	2	3	1	8	9	7
8	9	7	5	6	4	2	3	1
3	1	2	9	7	8	6	4	5
6	4	5	3	1	2	9	7	8
9	7	8	6	4	5	3	1	2

Solving harder sudoku's

This has been called "World's hardest sudoku" (see https://puzzling.stackexchange.com/a/389/7698) and we will solve it.

```r
sudokuTxt <- "
8 0 0  0 0 0  0 0 0
0 0 3  6 0 0  0 0 0
0 7 0  0 9 0  2 0 0
0 5 0  0 0 7  0 0 0
0 0 0  0 4 5  7 0 0
0 0 0  1 0 0  0 3 0
0 0 1  0 0 0  0 6 8
0 0 8  5 0 0  0 1 0
0 9 0  0 0 0  4 0 0"
sudoku <- as.matrix(
  read.table(text = sudokuTxt,
             col.names = letters[1:9]))
```

Solution to the “World’s hardest sudoku”

8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

Some benchmarking

Benchmarking using profvis is not so easy due to recursion and the number of function calls in zeallot but Rprof in base works fine:

```
Rprof(tmp <- tempfile())
solution <- solve(sudoku, findChoices)
Rprof()
summaryRprof(tmp)
```

and produces lots of output.

The operators %<-% and %>% are called a lot and bring a bit of overhead and we don't really need them. The code without is shown on the next slides.

In R code (2), removing %<-%

```
solve2 <- function(partialSolution, choicesFUN) {
  # Eliminate impossible values, give some suggestions
  # and flag contradictions.
  elStep <- eliminate(partialSolution, choicesFUN)
  # If dead end FALSE to trace back, if finished TRUE.
  if (elStep[[3]]) return(list(res = FALSE,
                               sol = NULL))
  if (all(elStep[[1]] %in% 1:9))
    return(list(res = TRUE, sol = elStep[[1]]))
  # Branching, exit when the solution is found.
  for (suggestion in elStep[[2]]) {
    ans <- solve2(suggestion, choicesFUN)
    if (ans$res) return(ans)
  }
  list(res = FALSE, sol = NULL)
}
```

In R code (2), removing %>%

The profiling also shows that the of the pipe operator slows the code down. It is not necessary either so we sacrifice some readability to get:

```
findChoices2 <- function(grid, i, j) {
  setdiff(setdiff(setdiff(1:9,
                          grid[i, ]),
                  grid[ , j]),
    grid[i - (i - 1) %% 3L + 0:2,
         j - (j - 1) %% 3L + 0:2])
}
```

OK, enough talk

show me a benchmark:

```
options(digits = 2)
microbenchmark::microbenchmark(
  pipe = solve(sudoku, findChoices),
  `no operators` = solve2(sudoku, findChoices2),
  times = 5)
```

```
## Unit: seconds
##          expr min   lq mean median   uq  max neval
##          pipe  37 37.1 37.2   37.2 37.2 37.2     5
##  no operators   5  5.1  5.1    5.1  5.1  5.2     5
```

This helps but can we do better?

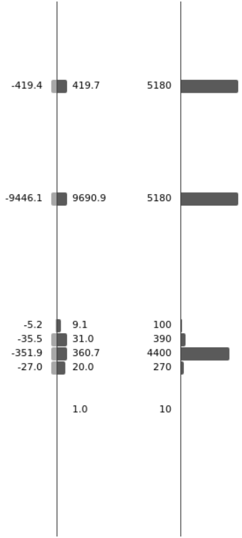
Let's do another profile

With the zeallot and pipe overhead gone, profiling works again. It shows the following:

```r
## ---- echo = TRUE-----------------------------------------------------
solve <- function(partialSolution, choicesFUN) {

  # Eliminate impossible values, give some suggestions
  # and flag contradictions.
  elStep <- eliminate(partialSolution, choicesFUN)
  # If dead end FALSE to trace back, if finshed TRUE.
  if (elStep[[3]]) return(list(res = FALSE,
                              sol = NULL))
  if (all(elStep[[1]] %in% 1:9))
    return(list(res = TRUE, sol = elStep[[1]]))
  # Branching, exit when the solution is found.
  for (suggestion in elStep[[2]]) {
    ans <- solve(suggestion, choicesFUN)
    if (ans$res) return(ans)
  }
  list(res = FALSE, sol = NULL)
}

## ---- echo = TRUE-----------------------------------------------------
eliminate <- function(grid, choicesFUN) {
  suggestions <- 0:9
  for (i in 1:nrow(grid)) { for (j in 1:ncol(grid)) {
    if (grid[i, j] == 0L) {
      choices <- choicesFUN(grid, i, j)
      if (length(choices) == 0L) {
        return(list(NULL, NULL, TRUE))
      } else if (length(choices) == 1L) {
        grid[i, j] <- choices
        return(list(grid, list(grid), FALSE))
      } else
        suggestions <- updateSuggestions(
          choices, grid, i, j, suggestions)
    }
  }}
  list(grid, suggestions, FALSE)
}
```



What now?

It seems hard to improve `findChoices2()` further but we can use C++. It is easier than you might expect.

C++ code

```
#include <Rcpp.h>
using namespace Rcpp;
IntegerMatrix subGrid(IntegerMatrix& x, int i, int j) {
  i -= i % 3; j -= j % 3;
  return x(Range(i, i + 2), Range(j, j + 2));
}
// [[Rcpp::export]]
IntegerVector findChoicesCpp(IntegerMatrix& x,
                             int i, int j) {
  IntegerVector candidates(9);
  std::iota(candidates.begin(), candidates.end(), 1);
  // C++ is zero-indexed.
  return setdiff(setdiff(setdiff(candidates,
          IntegerVector(x(i - 1, _))),
        IntegerVector(x(_, j - 1))),
      subGrid(x, i - 1, j - 1));
}
```

For comparison

```r
# Find all the choices allowed by the rules.
findChoices2 <- function(grid, i, j) {
  setdiff(setdiff(setdiff(1:9,
                    grid[i, ]),
            grid[ , j]),
    grid[i - (i - 1) %% 3L + 0:2,
         j - (j - 1) %% 3L + 0:2])
}
```

New benchmark

```r
options(digits = 2)
microbenchmark::microbenchmark(
  `no operators` = solve2(sudoku, findChoices2),
  `C++` = solve2(sudoku, findChoicesCpp),
  times = 10)
```

```
## Unit: milliseconds
##          expr  min   lq mean median   uq  max neval
##  no operators 4997 5010 5021   5025 5031 5037    10
##           C++  681  692  711    708  733  744    10
```

This is great, how do I get started

Simply

```r
install.packages('Rcpp')
```

and then start reading

```r
vignette('Rcpp-introduction')
```

Who is using it?

CRAN shows a lot packages

Reverse dependencies:

Reverse depends: ADMMsigma, Amelia, AnaCoDa, ASPBay, BaBooN, baitmet, Barycenter, bayesLopod, bcpa, beanz, bigReg, Biocomb, biofiles, blockmodels, bmlm, BMSC, breathteststan, brms, BuyseTest, CARBayes, CARBayesST, castor, ClusterStability, ClustMMDD, collUtils, ConConPiWiFun, copCAR, CoxPlus, cqrReg, cstab, ctsem, cytometree, datastructures, dbmss, DeLorean, DepthProc, dfpk, dils, DPP, DPWeibull, drgee, dslice, ecp, eggCounts, emIRT, erah, fastGHQuad, FDGcopulas, ffstream, FRESA.CAD, fugeR, gaston, GeDS, GENLIB, GenomicTools, glmgraph, glmmfields, Gmisc, gMWT, gppm, growcurves, growfunctions, GSE, GUTS, GWmodel, hBayesDM, hkevp, HUM, hypervolume, icenReg, idealstan, idem, IsingSampler, JAGUAR, jmotif, kergp, knor, landsepi, ldat, lm.br, ludic, MAINT.Data, ManifoldOptim, matchingR, mgwrsar, MixedDataImpute, mixlink, multicool, multinet, MVB, mvcluster, ngspatial, NlinTS, NPflow, OneArmPhaseTwoStudy, optimization, orQA, palm, PedCNV, PerMallows, PoweR, ProbitSpatial, propagate, prophet, quadrupen, RBesT, Rborist, RcppXts, recmap, regsem, remote, Rfast, RGeode, Rlda, Rmalschains, Rmixmod, RmixmodCombi, robustgam, rotations, RoughSets, RSNNS, Rsomoclu, rstanarm, rstansim, rtkore, SCPME, sdcTable, seismicRoll, sequences, SILGGM, simFrame, snipEM, spacodiR, spp, steadyICA, StMoSim, stpm, survHE, survSNP, synlik, tagcloud, tbart, themetagenomics, TLMoments, treatSens, treeclim, trialr, unmarked, vegclust, waffect, walker, wingui, wsrf

Reverse imports: ABCoptim, abcrf, AbsFilterGSEA, acc, accelerometry, acebayes, ACEt, acrt, AdaptiveSparsity, adeba, ADMM, ADMMnet, adpss, AhoCorasickTrie, AHR, alakazam, algstat, ALKr, alpaca, alphabetr, amt, anMC, ANN2, anomalyDetection, anytime, apcluster, ape, aphid, APML0, aricode, arrApply, ashr, attrCUSUM, augSIMEX, autoFRK, autothresholdr, autovarCore, BacArena, BalancedSampling, BAMBI, BAMMtools, BART, basad, batman, bayesAB, BayesBD, BayesComm, bayesDP, BayesESS, BayesFactor, BayesianTools, bayesImageS, bayeslm, bayesm, bayou, BCEE, BClustLonG, bcp, bcROCsurface, BCSub, bea.R, BEDMatrix, belg, benchr, BeSS, BeviMed, beyondWhittle, bfa, bfp, bgsmtr, BiDAG, bife, BIFIEsurvey, bigFastlm, bigKRLS, biglasso, bigmemory, BigQuic, bigreadr, bigrquery, bigstatsr, bigtime, BigVAR, bikedata, binaryGP, bindrcpp, binnednp, bio3d, bioacoustics, Bioi, biwavelet, blackbox, blockseg, blorr, BLPestimatoR, BLSM, bnclassify, bnnSurvival, BNPMIXcluster, BNSL, BoltzMM, BosonSampling, bpnreg, BradleyTerryScalable, BreedingSchemeLanguage, brlrmr, bsearchtools, bsplinePsd, bssm, btb, BTLLasso, BTR, BTYDplus, bvarsv, BVSNLP, bWGR, BWStest, carSurv, cartography, CaseBasedReasoning, CASMAP, catlearn, catSurv, cbinom, ccaPP, cccp, ccdrAlgorithm, cctools, CDM, cellWise, CENFA, CFC, cgAUC, changepointsHD, ChannelAttribution, chopthin, chunkR, circglmbayes, CircularDDM, circumplex, cIRT, CKLRT, cladoRcpp, clampSeg, classifierplots, classify, cld2, cld3, cleanEHR, clere, climdex.pcic, ClinicalTrialSummary, cliqueMS, clogitboost, clusrank, ClusterR, clusteval, ClustVarLV, ClusVis, CMF, cna, CNull, coala, coda.base, cOde, coga, combiter, comperank, CompGLM, coneproj, contoureR, cord, corpustools, CorReg, Countr, covTestR, CovTools, cpr, crawl, creditr, ctmcd, cubature, curstatCI, cutpointr, CVR, CVXR, cycleRtools, Cyclops, DatabionicSwarm, datafsm, DataGraph, DatAssim, DataVisualizations, dbscan, dcurver, ddalpha, DDPGPSurv, DDRTree, decido, deepboost, demu, densityClust, DEploid, DescTools, detectRUNS, detrendr, dexterMST, dfphase1, dggridR, DGM, dHSIC, diagis, diceR, DiffNet, diffrprojects, DiffusionRgqd, DiffusionRimp, DiffusionRjgqd, diffusr, dina, disclapmix, discretecdAlgorithm, diversitree, diveRsity, divest, dlib, DNAtools, dng, dodgr, dplyr, dqrng, DrImpute, DstarM, DStree, DtD, dtwclust, dvmisc, dynamichazard, dynamo, dynfrail, dynsbm, dynutils, EAinference, easyVerification, EBMAforecast, ECctmc, ECFsup, eDMA, eive, elmNNRcpp, elo, EloChoice, EMbC, Emcdf, EMMIXgene, EMVS, energy, ensembleEN, epiphy, episode, equateMultiple, eseis, ESGtoolkit, esreg, essHist, EstHer, estimatr, estudy2, ETAS, etm, eulerr, evolqg, EWGoF, exif, extraDistr, exuber, facilitation, factorcpt, factorstochvol, FarmSelect, FarmTest, fastAdaboost, FastBandChol, fastcmh, fasteraster, fasterElasticNet, fasterize, FastGP, fastJT, fastLink, fastM, fastrtext, FastSF, fastTextR, FBFsearch, fbroc, fdadensity, fdaMixed, fdapace, fdasrvf, fDMA, FDRreg, FDRSeg, feather, FeatureHashing, FENmlm, FIACH, fICA, filesstrings, filling, fingerPro, FIT, flam, flan, flars, flexsurv, flock, FLSSS, forecast, forecastSNSTS, forestControl, forestFloor, fourierin, fourPNO, fpeek, fractional, frailtyEM, FRegSigCom, FRK, fromo, fs, FSelectorRcpp, FSInteract, fst, FunChisq, Funclustering, FUNLDA, fuser, futureheatwaves, fwsim, GA, GADAG, gapfill, GAS, gaselect, gastempt, GauPro, GBJ, gbp, gcKrig, GCPM, GDINA, gdm, gdpc, gdtools, gee4, GEEaSPU, geiger, genepop, GeneralizedUmatrix, genie, GenomicMating, geoCount, geogrid, geohash, geojsonR, geojsonsf, geoops, GERGM, ggforce, GGIR, ggraph, ggrepel, GiRaF, gjam, gkmSVM, glamlasso, glcm, glm.deploy, GLMaSPU, glmBfp, glmmsr, GMCM, Gmedian, goldi, googlePolylines, GPareto, GPCMlasso, GPGame, GpGp, gpuR, GPvam, gRain, graphicalVAR, graphkernels, graphql, grattan, gRbase, GreedyEPL, GreedySBTM, gren, grf, gRim, groupedSurv, grove, GrpString, gscounts, gsEasy, gsynth, GUILDS, gwfa, GxM, h5, HardyWeinberg, hashmap, haven, hawkes, hdbinseg, HDclust, hdme, hesim, hetGP, HHG, hierarchicalSets, hipread, HistDAWass, hit, HMMEsolver, hogsvdR, hommel, HSAR, hsphase, htdp, htmltidy, htmltools, httpuv, humaniformat, humanleague, hunspell, hyper2, hyphenatr, IAPWS95, iBATCGH, ibm, ibmcraftr, iBST, icamix, ICAOD, ICcalib, iccbeta, icd, icensmis, icosa, icr, ICRanks, icRSF, ICtest, IHSEP, iilasso, ijtiff, imager, imagerExtra, imagine, imbalance, immer, imp4p, imptree, imputeMulti, imputeTS, inarmix, inca, IncDTW, incgraph, indelmiss, iNextPD, inferr, inplace, intcensROC, IntegratedMRF, interflex, interp, IntervalSurgeon, ipft, iprior, iptools, ipumsr, iRF, irregulAR1, IRTpp, ISOpureR, IsoSpecR, isqg, jaccard, jiebaR, JMbayes, jmcm, JMcmprsk, jmvconnect, joineRML, joinXL, JOUSBoost, JSM, jtGWAS, JuliaCall, JumpTest, junctions, JuniperKernel, jwutil, kamila, kde1d, kdecopula, kdevine, kdtools, kernDeepStackNet, kernelboot, KernelKnn, KernSmoothIRT, KeyboardSimulator, kmcudaR, kmer, kohonen, KoulMde, KRIG, KSgeneral, kvh, l0ara, L0Learn, LaF, LAM, LambertW, lamW, LANDD, landscapemetrics, landscapeR, Langevin, LassoBacktracking, later, lbfgs, LBSPR, lclGWAS, lcopula, lexRankr, lfl, libstableR, lidR, lifecontingencies, liger, lime, linearQ, LocalControl, logKDE, lowmemtkmeans, LowWAFOMNX, LowWAFOMSobol, lpirfs, lpme, lsbclust, lslx, lubridate, lucr, lvec, lwgeom, MADPop, mafs, magick, mapview, marked, markophylo, markovchain, MAT, matchingMarkets, MatchItSE, mateable, MatrixCorrelation, MatrixLDA, MAVE, maxent, mbbefd, mbgraphic, mcemGLM, mcga, mclustcomp, MCMCprecision, mcmcse, mcPAFit, mdmb, meaRtools, medfate, MediaK, meltt, MESS, metacoder, metafolio, MetaheuristicFPA, meteoland, mets, mev, mfGARCH, mice, miceadds, miceFast, microclass, micropan, microsamplingDesign, microseq, miic, milr, minimaxdesign, minqa, miRNAss, mirt, mirtCAT, miscset, MiSPU, missDeaths, mixdir, mixedMem, mixpack, mixR, mizer, mkde, mlmc, mlvocab, mlxR, MM4LMM, mmand, mmapcharr, MMPPsampler, mnis, mobsim, ModelMetrics, momentuHMM, Morpho, mosum, mousetrap, move, moveHMM, moveWindSpeed, mp, MPTinR, mrbsizeR, mrgsolve, MRS, msde, MSGARCH, MTS, MuChPoint, multdyn, MultiBD, MultiFit, multinets, multistateutils, MultivariateRandomForest, mumm, mvabund, mvnfast, mwaved, myTAI, n1qn1, nabor, NAM, nandb, ncpen, NCutYX, ndjson, NestedCategBayesImpute, netcoh, netdiffuseR, netrankr, NetRep, NetworkDistance, NetworkInference, networkR, neuroim, nlmixr, nmfgpu4R, nmixgof, nmslibR, nngeo, NNLM, noncompliance, nonlinearTseries, nprobust, nse, Numero, nVennR, odbc, odeintr, odpc, oem, officer, olctools, olsrr, ompr, onemap, onlinePCA, ontologySimilarity, openair, openCR, OpenImageR, OpenMx, openxlsx, optiSel, optiSolve, optmatch, opusminer, Orcs, ordinalClust, ordinalForest, orthoDr, osmdata, osqp, outbreaker2, PAC, packcircles, pacotest, padr, PAFit, pander, PanelCount, parallelDist, partialAR, partialCI, particles, patternplot, pcadapt, pcalg, pccc, pcIRT, PCMRS, pdftools, pdSpecEst, pedometrics, penalized, PenCoxFrail, penMSM, pense, Peptides, perccal, pgdraw, pgee.mixed, pGPx, ph2bayes, ph2bye, phangorn, phantom, Phase123, PhenotypeSimulator, philentropy, phonics, phybreak, phylobase, phylocurve, PhylogeneticEM, phylosignal, physiology, PieceExpIntensity, Pijavski, pimeta, pinbasic, pirate, piton, PKPDmisc, plac, planar, planor, PLMIX, plotSEMM, plyr, poisbinom, poisDoubleSamp, polyfreqs, polyRAD, polysat, polywog, POMaSPU, popkin, PP, PPforest, PPtreeViz, PQLseq, prclust, precrec, predkmeans, PReMiuM, primes, PRIMME, princurve, pROC, prodlim, ProjectionBasedClustering, projmanr, projpred, promises, propr, prospectr, protolite, prototest, protViz, pryr, psd, pseudorank, psgp, PUlasso, puniform, purrrlyr, PWD, qGaussian, qrencoder, QRM, QTL.gCIMapping, QTL.gCIMapping.GUI, qualpalr, quanteda, quantregRanger, quantspec, QuantTools, queuecomputer, qVarSel, qwraps2, radiomics, ramcmc, randomUniformForest, rangeBuilder, ranger, Rankcluster, rankdist, RaPKod, rare, rarhsmm, raster, ratematrix, rbacon, Rblpapi, RClickhouse, RcppAlgos, RcppAnnoy, RcppAPT, RcppArmadillo, RcppBDT, RcppBlaze, RcppCCTZ, RcppClassic, RcppClassicExamples, RcppCNPy, RcppCWB, RcppDE, RcppDL, RcppEigen, RcppEigenAD, RcppExamples, RcppFaddeeva, RcppGetconf, RcppGreedySetCover, RcppGSL, RcppHMM, RcppHoney, RcppMeCab, RcppMLPACK, RcppMsgPack, RcppNumerical, RcppQuantuccia, RcppRedis, RcppRoll, RcppSMC, RcppStreams, RcppThread, RcppTN, RcppTOML, RcppXPtrUtils, RcppZiggurat, rcss, Rdimtools, rdist, rDotNet, Rdtq, re2r, readobj, readr, readstata13, readxl, RealVAMS, recexcavAAR, reclin, reconstructr, recosystem, recurse, reda, redist, rEDM, refinr, regnet, ReIns, relSim, rem, rENA, ReorderCluster, repfdr, repolr, reproducible, resemble, reshape2, reticulate, revdbayes, rexpokit, rforensicbatwing, rFTRLProximal, rgam, rgeolocate, RI2by2, RInside, Rip46, ripa, rIsing, riskRegression, rivr, RJafroc, rkvo, Rlabkey, rlas,

but I believe a lot of Rcpp never makes it to CRAN.

Questions?

Any questions, remarks or observations?

On GitHub

You can find this presentation and related materials on

https://github.com/bobjansen/RcppSudoku