nested-design-v6

April 11, 2020

1 Nesting Designs, version 6

1.1 Load packages.

[1]:
```
require(abind)
require(data.table)
require(deSolve)
require(magrittr)
require(rTensor)

require(ggplot2)
require(GGally)
```

```
Loading required package: abind
Loading required package: data.table
Loading required package: deSolve
Loading required package: magrittr
Loading required package: rTensor
Loading required package: ggplot2
Loading required package: GGally
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```

1.2 Function to generate simulations.

[2]:
```
# Create a multivariate function with specified properties:
#   tmax: maximum time
#   multiplicities: number of correlations each parameter has
#   degrees: polynomial degree of each parameter
#   dimension: the dimension of the output
#   returns a multivariate function of the vector of parameters and time
makeGenerator <- function(multiplicities, degrees, dimension) {

    single <- function(degree) {
      x0 <- runif(1)
      z0 <- runif(1)
```

```
      function(x) {
          if (x < x0)
              0
          else
              z0 * (x - x0)^degree
      }
    }

    locations <- lapply(multiplicities, function(m) sample(1:dimension, m))
    functions <- lapply(degrees, single)

    start <- runif(dimension, -0.25, 0.75)
    coefs <- matrix(runif(dimension^2, -0.25, 0.75), dimension, dimension)
    shift <- matrix(runif(dimension^2, -0.25, 0.75), dimension, dimension)

    function(x, ts) {
        z <- rep(0, dimension)
        for (i in 1:length(locations))
            for (j in locations[[i]])
                z[j] <- z[j] + functions[[i]](x[i])
        ode(start, ts, function(t, y, params) {list((coefs %*% y) * z * (1 -␣
 ↪((shift %*% y) * z)))})
    }

}
```

1.3 Functions for nesting designs

[3]:
```
ssa.digits <- 30
```

[4]:
```
ssa.scale <- 2^ssa.digits
```

[5]:
```
ssa.level <- function(i) {
    if (i == 0)
        return(ssa.digits)
    n <- 0
    while (bitwAnd(i, 1) == 0) {
        n <- n + 1
        i <- bitwShiftR(i, 1)
    }
    n
}
```

[6]:
```
ssa.depth <- function(i) {
    ssa.digits - ssa.level(i)
}
```

[7]:
```
ssa.corner <- function(i, offset=0){
    d <- ssa.level(i) - offset
    c(i - 2^d, i + 2^d)
}
```

[8]:
```
ssa.corners <- function(i1, i2, i3, offset=0) {
    axis <- function(s1, s2, s3) {
        if (s1 == 0)
            4 * s1 + 2 * s2 + s3
        else
            4 * (1 - s1) + 2 * (1 - s2) + (1 - s3)
    }
    merge(merge(merge(
        data.table(
            d1 = 1,
            d2 = 2,
            d3 = 4
        ),
        data.table(d1=1, i1=ssa.corner(i1, offset=offset), s1=0:1), by="d1",␣
 ↪allow.cartesian=TRUE),
        data.table(d2=2, i2=ssa.corner(i2, offset=offset), s2=0:1), by="d2",␣
 ↪allow.cartesian=TRUE),
        data.table(d3=4, i3=ssa.corner(i3, offset=offset), s3=0:1), by="d3",␣
 ↪allow.cartesian=TRUE
    )[,
    .(
        i1,
        i2,
        i3,
        axis=mapply(axis, s1, s2, s3)
    )]
}
```

[9]:
```
ssa.candidates <- function(i1, i2, i3) {
    CJ(
        i1=c(i1, ssa.corner(i1)),
        i2=c(i2, ssa.corner(i2)),
        i3=c(i3, ssa.corner(i3))
    )[,
        .(
            i1,
            i2,
            i3,
            x1 = i1 / ssa.scale,
            x2 = i2 / ssa.scale,
            x3 = i3 / ssa.scale
        )
    ]
```

```
}
```

[10]:
```
ssa.start <- function() {
    result <- CJ(
        i1=c(0, ssa.scale),
        i2=c(0, ssa.scale),
        i3=c(0, ssa.scale)
    )[,
        .(
            sequence = 1:8     ,
            generation = 0     ,
            i1                 ,
            i2                 ,
            i3                 ,
            x1 = i1 / ssa.scale,
            x2 = i2 / ssa.scale,
            x3 = i3 / ssa.scale,
            compute = TRUE     ,
            measure = FALSE    ,
            probed  = TRUE     ,
            s1 = 0             ,
            s2 = 0             ,
            s3 = 0             ,
            s  = 0
        )
    ]
    result <- rbind(
        result,
        data.table(
            sequence = 9       ,
            generation = NA    ,
            i1 = ssa.scale / 2,
            i2 = ssa.scale / 2,
            i3 = ssa.scale / 2,
            x1 = 0.5           ,
            x2 = 0.5           ,
            x3 = 0.5           ,
            compute = FALSE    ,
            measure = TRUE     ,
            probed  = FALSE    ,
            s1 = 0. / 0.       ,
            s2 = 0. / 0.       ,
            s3 = 0. / 0.       ,
            s  = 0. / 0.
        )
    )
    setkeyv(result, c("i1", "i2", "i3"))
```

```
}
```

[11]:
```
ssa.compute <- function(f, ts, xs, ys) {
    needed <- xs[compute == TRUE]
    for (row in 1:nrow(xs)) {
        ys <- rbind(
            ys,
            data.table(f(as.numeric(xs[row, .(x1, x2, x3)]), ts))[,
                .(
                    sequence = xs[row, sequence],
                    t = time,
                    y1 = `1`,
                    y2 = `2`,
                    y3 = `3`
                )
            ]
        )
    }
    xs[compute == TRUE, compute := FALSE]
    ys
}
```

[12]:
```
ssa.measure <- function(xs, ys, focus=2) {

    result <- merge(
        merge(
            xs[measure == TRUE][, ssa.corners(i1, i2, i3), by=.
 ↪(center=sequence)],
            xs,
            by=c("i1", "i2", "i3"),
            allow.cartesian=TRUE
        )[, .(center, axis, sequence)],
        ys,
        by="sequence",
        allow.cartesian=TRUE
    )[,
        .(y1 = mean(y1), y2 = mean(y2), y3 = mean(y3)), by=.(sequence=center,␣
 ↪axis, t)
    ][,
        .(s1 = sd(y1), s2 = sd(y2), s3 = sd(y3)), by=.(sequence, t)
    ][,
        .(s1 = max(s1), s2 = max(s2), s3 = max(s3)), by=sequence
    ]

    xs[
        sequence %in% result$sequence,
        `:=`(
            s1=result$s1,
```

```
            s2=result$s2,
            s3=result$s3,
            generation=xs[, 1 + max(generation, na.rm=TRUE)]
        )
    ]

    y1min = min(ys$y1)
    y1max = max(ys$y1)
    y2min = min(ys$y2)
    y2max = max(ys$y2)
    y3min = min(ys$y3)
    y3max = max(ys$y3)

    normalize <- function (i, s1, s2, s3)
        max(s1 / (y1max - y1min), s2 / (y2max - y2min), s3 / (y3max - y3min)) *␣
 ↪focus^ssa.depth(i)
    xs[, `:=`(measure = FALSE, s = mapply(normalize, i1, s1, s2, s3))]

    result
}
```

[13]:
```
ssa.probe <- function(xs) {
    probe <- xs[
        probed == FALSE,
        .(sequence, i1, i2, i3, rank=frank(-s, ties.method="random"))
    ][
        rank == 1,
        .(sequence, i1, i2, i3)
    ]
    candidates <- ssa.candidates(probe$i1, probe$i2, probe$i3)
    setkeyv(candidates, c("i1", "i2", "i3"))
    candidates <- candidates[!xs]
    candidates <- candidates[, .(
        sequence = xs[, max(sequence)] + (1:nrow(candidates)),
        generation = NA,
        i1, i2, i3,
        x1, x2, x3,
        compute = TRUE ,
        measure = FALSE,
        probed  = FALSE,
        s1 = 0. / 0.,
        s2 = 0. / 0.,
        s3 = 0. / 0.,
        s  = 0. / 0.
    )]
    probes <- ssa.corners(probe$i1, probe$i2, probe$i3, offset=1)
    probes <- probes[, .(
```

```
        sequence = max(candidates$sequence) + (1:nrow(probes)),
        generation = NA,
        i1, i2, i3,
        x1 = i1 / ssa.scale,
        x2 = i2 / ssa.scale,
        x3 = i3 / ssa.scale,
        compute = FALSE,
        measure = TRUE ,
        probed  = FALSE,
        s1 = 0. / 0.,
        s2 = 0. / 0.,
        s3 = 0. / 0.,
        s  = 0. / 0.
    )]
    result <- rbind(
        xs,
        candidates,
        probes
    )
    result[sequence == probe$sequence, `:=`(compute=TRUE, measure=FALSE,␣
↪probed=TRUE)]
    setkeyv(result, c("i1", "i2", "i3"))
    list(
        sequence=probe$sequence,
        xs=result
    )
}
```

1.4 Reproducible random numbers.

[14]:
```
RNGkind("Mersenne-Twister", "Inversion", "Rejection")
```

[15]:
```
set.seed(46)
```

1.5 Create a simulation function.

[16]:
```
f <- makeGenerator(c(2, 2, 3), c(0, 1, 2), 3)
```

1.6 Example application

1.6.1 Set time resolution.

[17]:
```
ts <- (0:20) / 2
```

1.6.2 Initialize experiment at bounds of domain.

[18]:
```
xs <- ssa.start()
```

[19]:
```
xs
```

A data.table: 9 x 15

sequence <dbl>	generation <dbl>	i1 <dbl>	i2 <dbl>	i3 <dbl>	x1 <dbl>	x2 <dbl>	x3 <dbl
1	0	0	0	0	0.0	0.0	0.0
2	0	0	0	1073741824	0.0	0.0	1.0
3	0	0	1073741824	0	0.0	1.0	0.0
4	0	0	1073741824	1073741824	0.0	1.0	1.0
9	NA	536870912	536870912	536870912	0.5	0.5	0.5
5	0	1073741824	0	0	1.0	0.0	0.0
6	0	1073741824	0	1073741824	1.0	0.0	1.0
7	0	1073741824	1073741824	0	1.0	1.0	0.0
8	0	1073741824	1073741824	1073741824	1.0	1.0	1.0

1.6.3 Iterate several times.

[20]:
```
ys <- NULL
sequences <- NULL
```

[21]:
```
for (k in 1:250) {
    # Perform pending computations.
    ys <- ssa.compute(f, ts, xs, NULL)
    # Measure divergences.
    ssa.measure(xs, ys, focus = (sqrt(5) + 1) / 2)
    # Select where to probe further.
    result <- ssa.probe(xs)
    # Tally results.
    sequences <- rbind(
        sequences,
        xs[
            sequence == result$sequence,
            .(
                sequence,
                x1, x2, x3,
                generation,
                depth=mapply(ssa.depth, i1))
        ]
    )
    # Update grid.
    xs <- result$xs
}
sequences
```

sequence <dbl>	x1 <dbl>	x2 <dbl>	x3 <dbl>	generation <dbl>	depth <dbl>
9	0.500000	0.500000	0.500000	1	1
33	0.750000	0.250000	0.750000	2	2
55	0.625000	0.125000	0.875000	3	3
56	0.625000	0.375000	0.625000	3	3
109	0.687500	0.312500	0.562500	5	4
108	0.562500	0.437500	0.687500	5	4
35	0.750000	0.750000	0.750000	2	2
179	0.625000	0.625000	0.875000	8	3
181	0.625000	0.875000	0.875000	8	3
105	0.562500	0.312500	0.562500	5	4
107	0.562500	0.437500	0.562500	5	4
158	0.531250	0.406250	0.718750	7	5
163	0.593750	0.468750	0.656250	7	5
268	0.593750	0.468750	0.593750	12	5
245	0.531250	0.281250	0.593750	11	5
250	0.593750	0.343750	0.531250	11	5
262	0.531250	0.406250	0.593750	12	5
162	0.593750	0.406250	0.718750	7	5
164	0.593750	0.468750	0.718750	7	5
208	0.687500	0.562500	0.937500	9	4
82	0.562500	0.187500	0.812500	4	4
226	0.562500	0.937500	0.812500	10	4
83	0.562500	0.187500	0.937500	4	4
205	0.562500	0.687500	0.812500	9	4
316	0.578125	0.484375	0.671875	14	6
315	0.578125	0.484375	0.640625	14	6
58	0.875000	0.125000	0.625000	3	3
57	0.625000	0.375000	0.875000	3	3
659	0.562500	0.437500	0.812500	29	4
661	0.687500	0.312500	0.812500	29	4
...	...	...	...	...	...
1672	0.5703125	0.9296875	0.6171875	75	7
1670	0.5703125	0.9140625	0.6171875	75	7
2093	0.5703125	0.7578125	0.6171875	93	7
2150	0.5859375	0.4609375	0.6171875	96	7
2135	0.5859375	0.8984375	0.6171875	95	7
2267	0.5859375	0.2578125	0.6171875	102	7
2291	0.5859375	0.8203125	0.6171875	103	7
2246	0.5859375	0.3203125	0.6171875	101	7
2144	0.5703125	0.4453125	0.6171875	96	7
2363	0.5781250	0.9843750	0.7968750	107	6
1871	0.5859375	0.4140625	0.7109375	84	7
2734	0.5703125	0.4921875	0.7109375	126	7
2735	0.5859375	0.4765625	0.6953125	126	7
1872	0.5859375	0.4296875	0.6953125	84	7
2820	0.5859375	0.9921875	0.6953125	130	7
2835	0.5859375	0.9765625	0.5390625	131	7
2834	0.5703125	0.9921875	0.5546875	131	7
2698	0.5859375	0.9453125	0.5390625	124	7
2630	0.5312500	0.7812500	0.8437500	121	5
2514	0.5312500	0.2812500	0.8437500	115	5

A data.table: 250 x 6

1.6.4 Save results.

[22]:
```
write.table(xs, file="xs-v6.csv", row.names=FALSE, sep=",")
xs %>% dim
```

1. 5509 2. 15

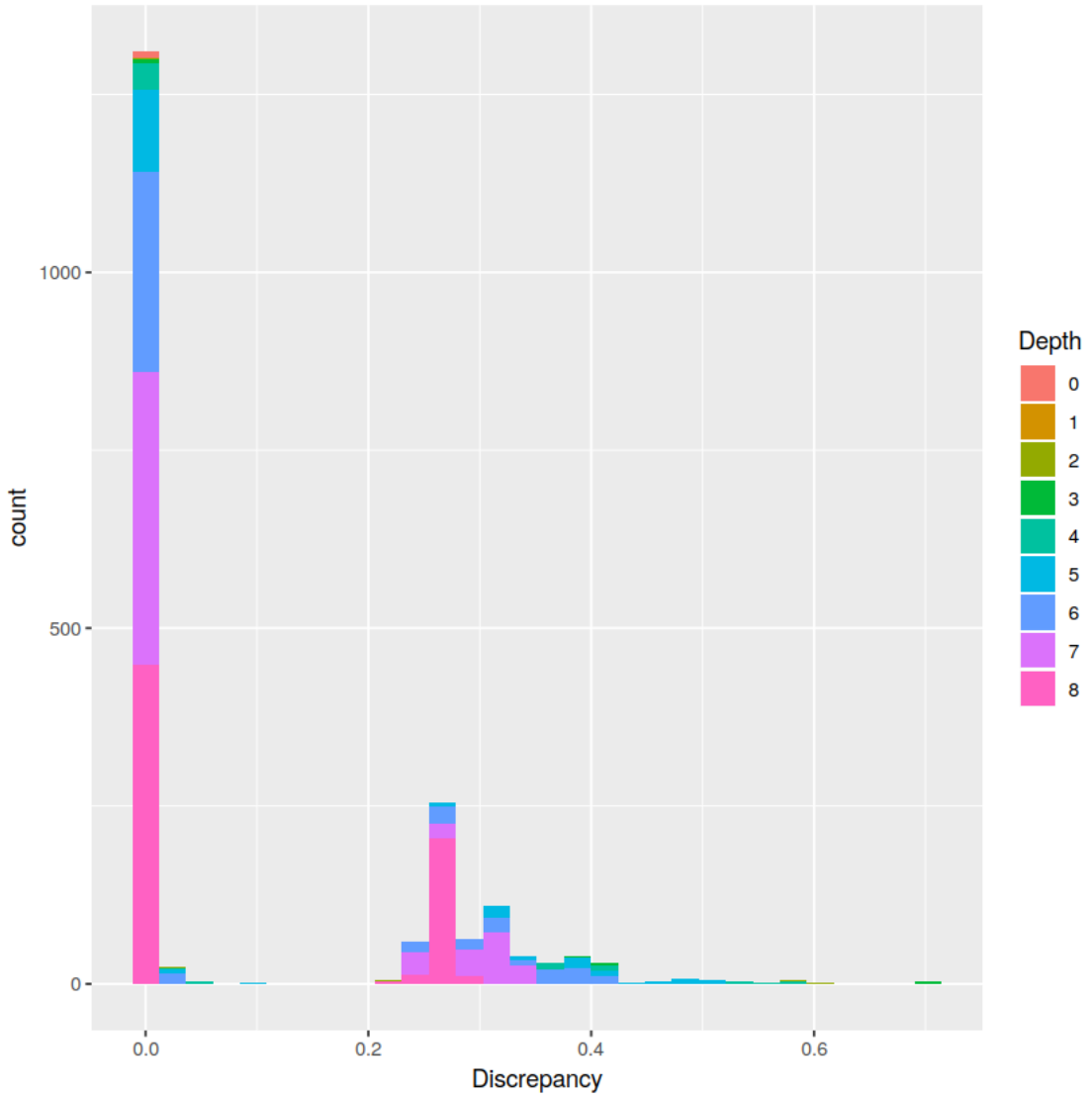
[23]:
```
write.table(ys, file="ys-v6.csv", row.names=FALSE, sep=",")
ys %>% dim
```

1. 115164 2. 5

1.6.5 Plot sampling pattern.

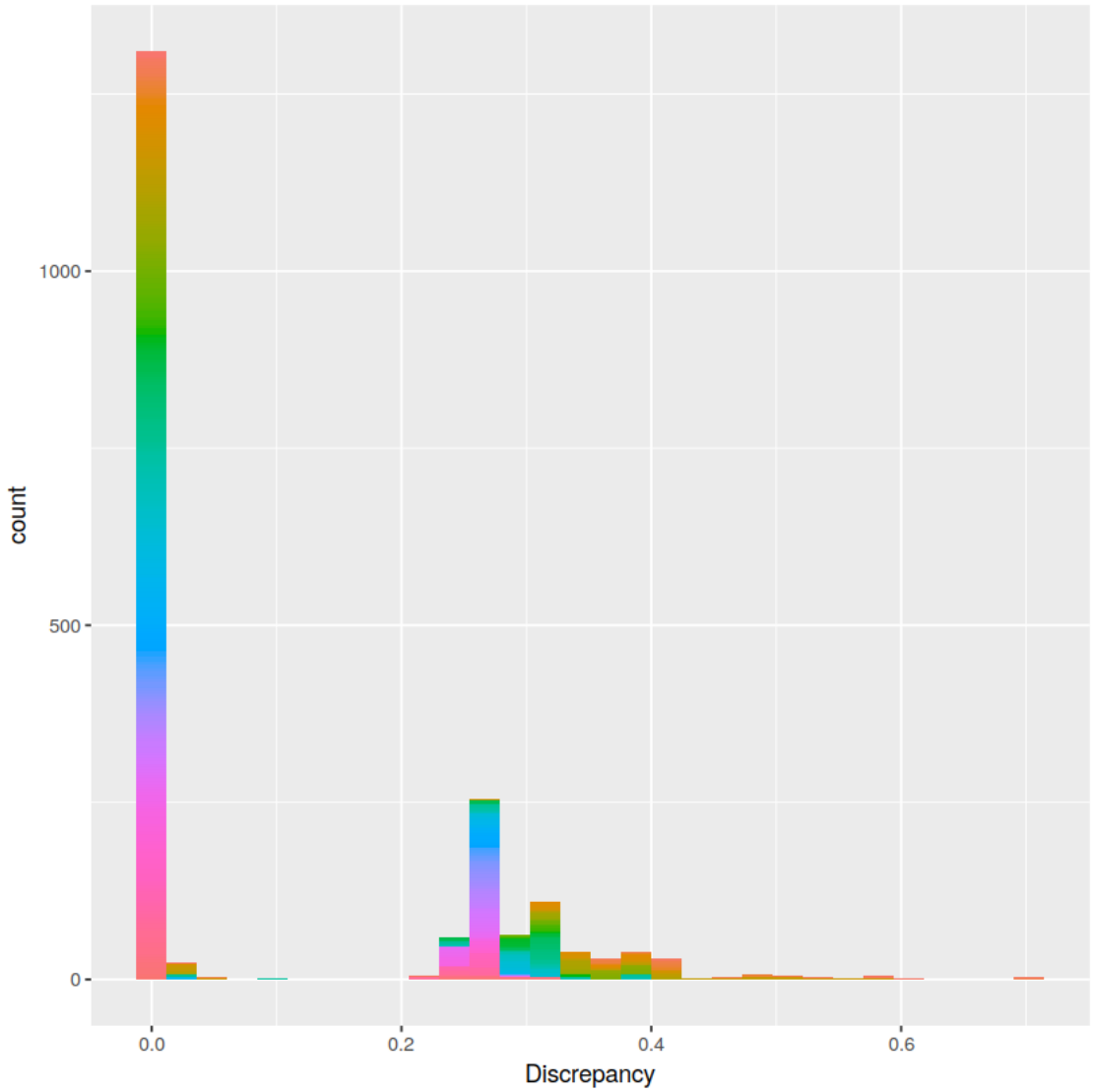
Distribution of discrepancies, distinguished by depth.

[24]:
```
ggplot(
    xs[!is.na(generation), .(Generation=factor(generation), Discrepancy=s,␣
 ↪Depth=factor(mapply(ssa.depth, i1)))],
    aes(x=Discrepancy, fill=Depth)
) + geom_histogram(bins = 30)
```



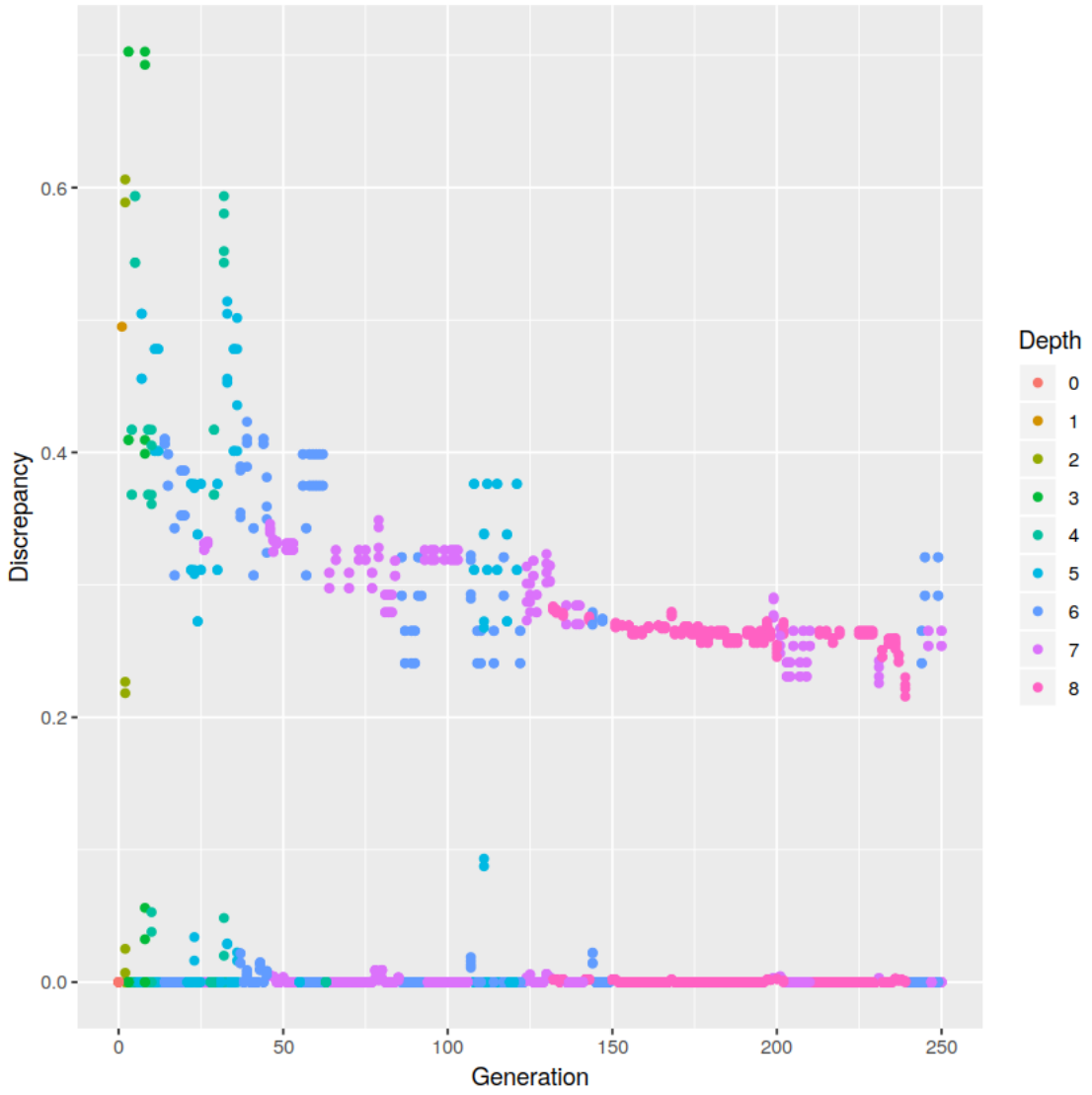
Distribution of discrepancies, distinguished by generation.

[32]:
```
ggplot(
    xs[!is.na(generation), .(Generation=factor(generation), Discrepancy=s,␣
 ↪Depth=factor(mapply(ssa.depth, i1)))],
    aes(x=Discrepancy, fill=Generation)
) + geom_histogram(bins = 30)  + theme(legend.position = "none")
```



Discrepancies and depth as a function of generation.

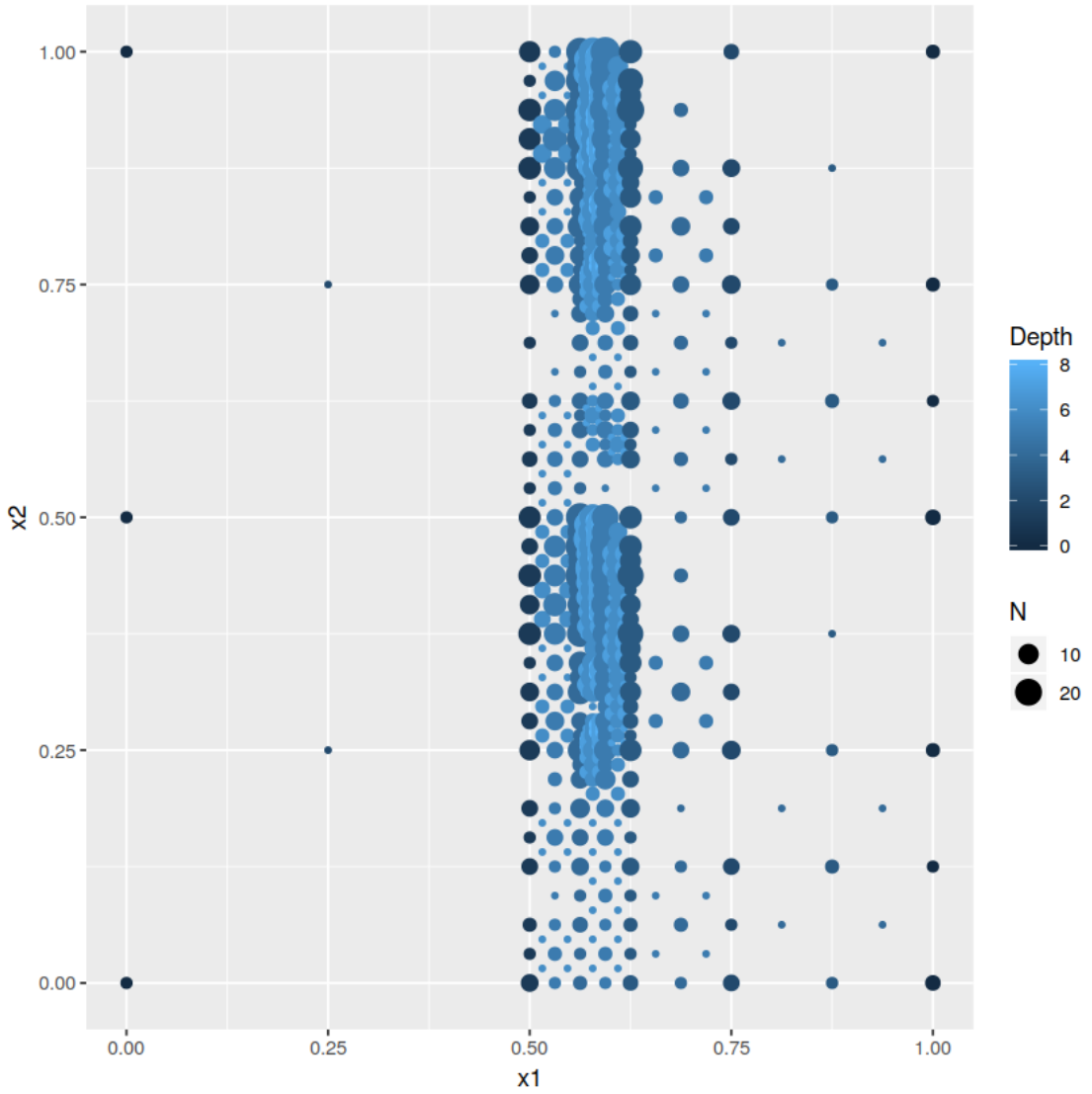
[26]:
```
ggplot(
    xs[!is.na(generation), .(Generation=generation, Discrepancy=s,␣
 ↪Depth=factor(mapply(ssa.depth, i1)))],
    aes(x=Generation, y=Discrepancy, color=Depth)
) + geom_point()
```



Marginal densities of sampling.

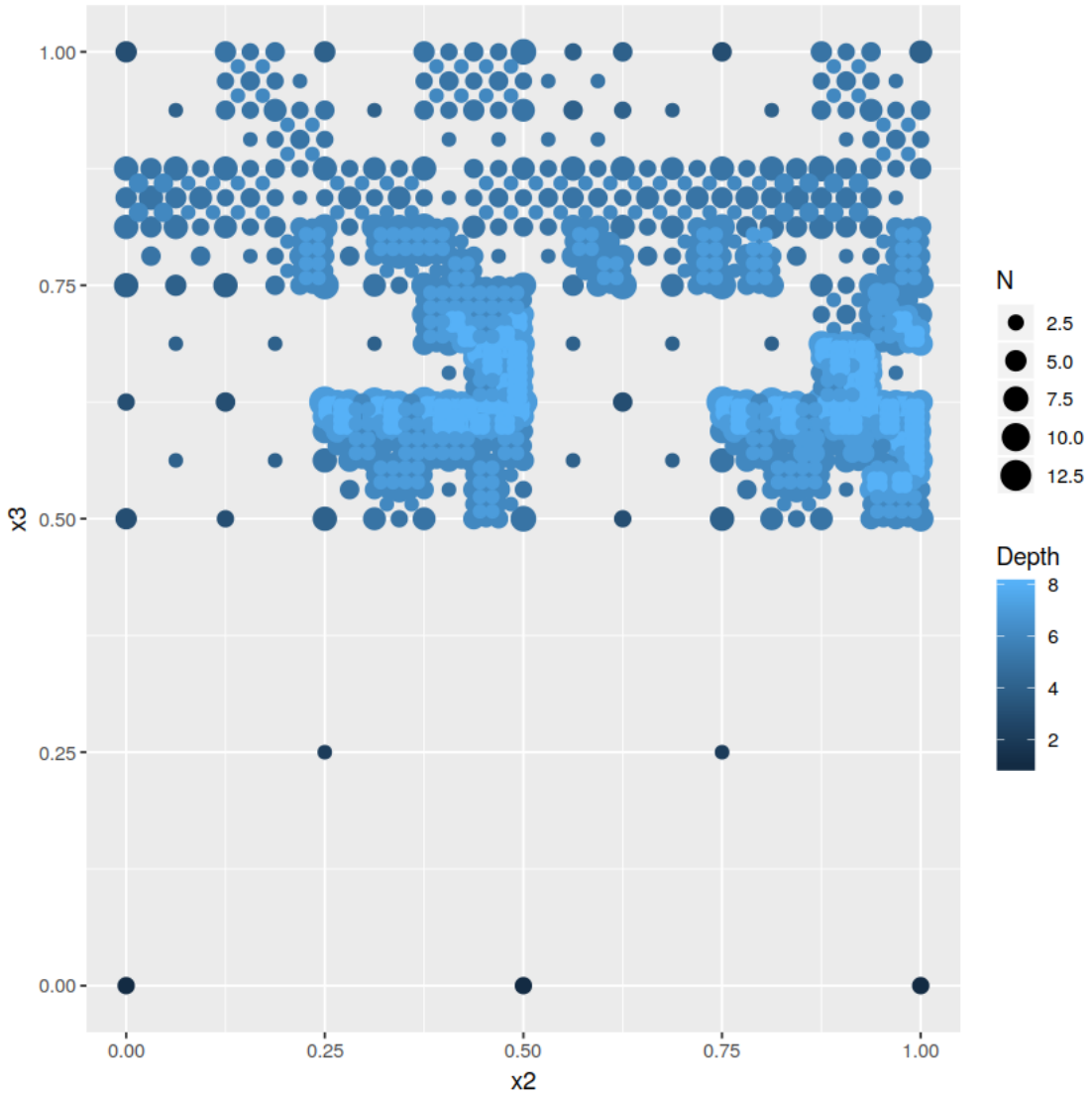
x1 vs x2.

[27]:
```
ggplot(
    xs[compute == FALSE, .(N=.N, Depth=max(mapply(ssa.depth, i1))), by=.(x1,␣
 ↪x2)],
    aes(x=x1, y=x2, size=N, color=Depth)
) + geom_point()
```



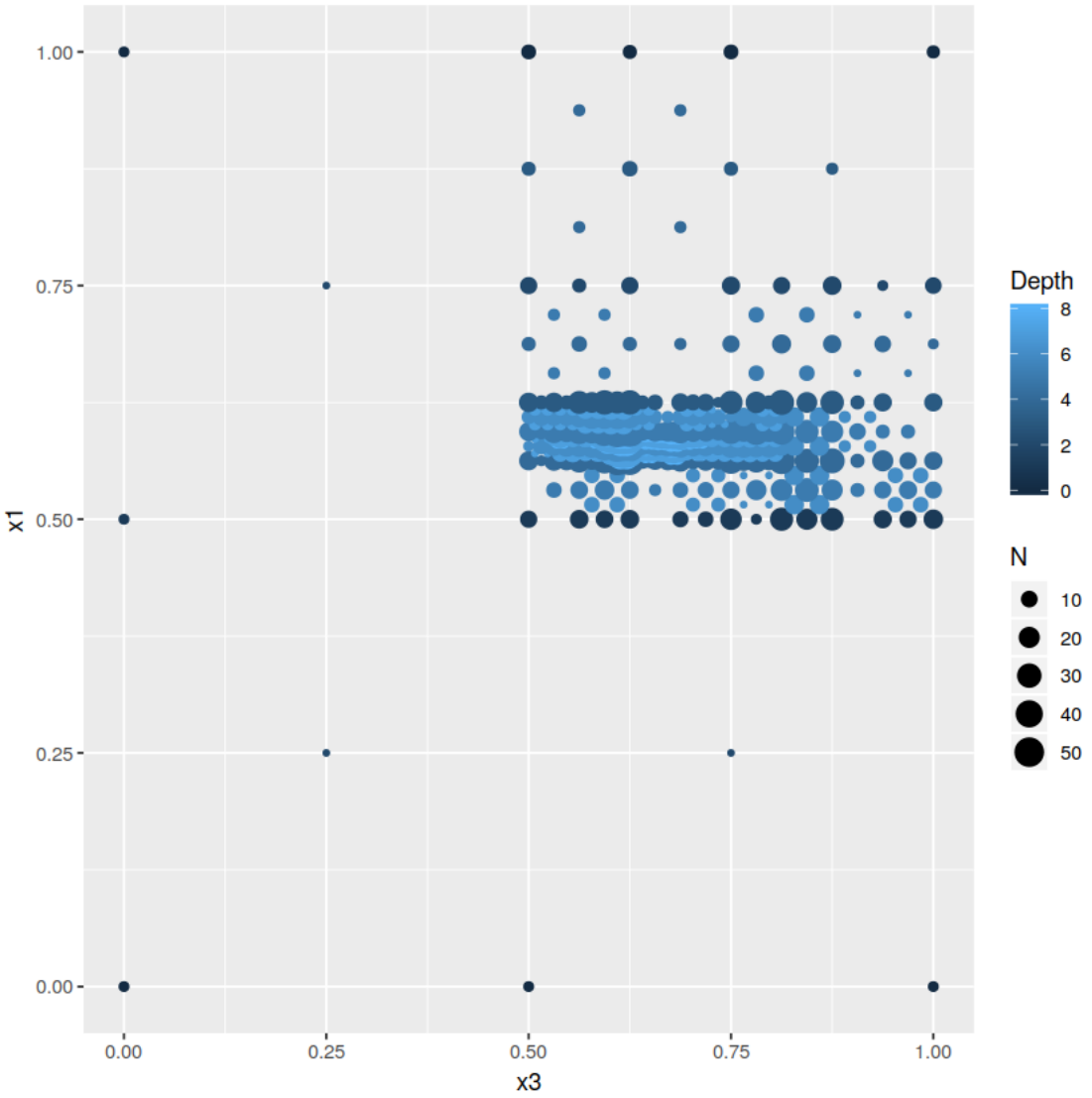
x2 vs x3.

[28]:
```
ggplot(
    xs[compute == FALSE, .(N=.N, Depth=max(mapply(ssa.depth, i1))), by=.(x2,␣
 ↪x3)],
    aes(x=x2, y=x3, size=N, color=Depth)
) + geom_point()
```



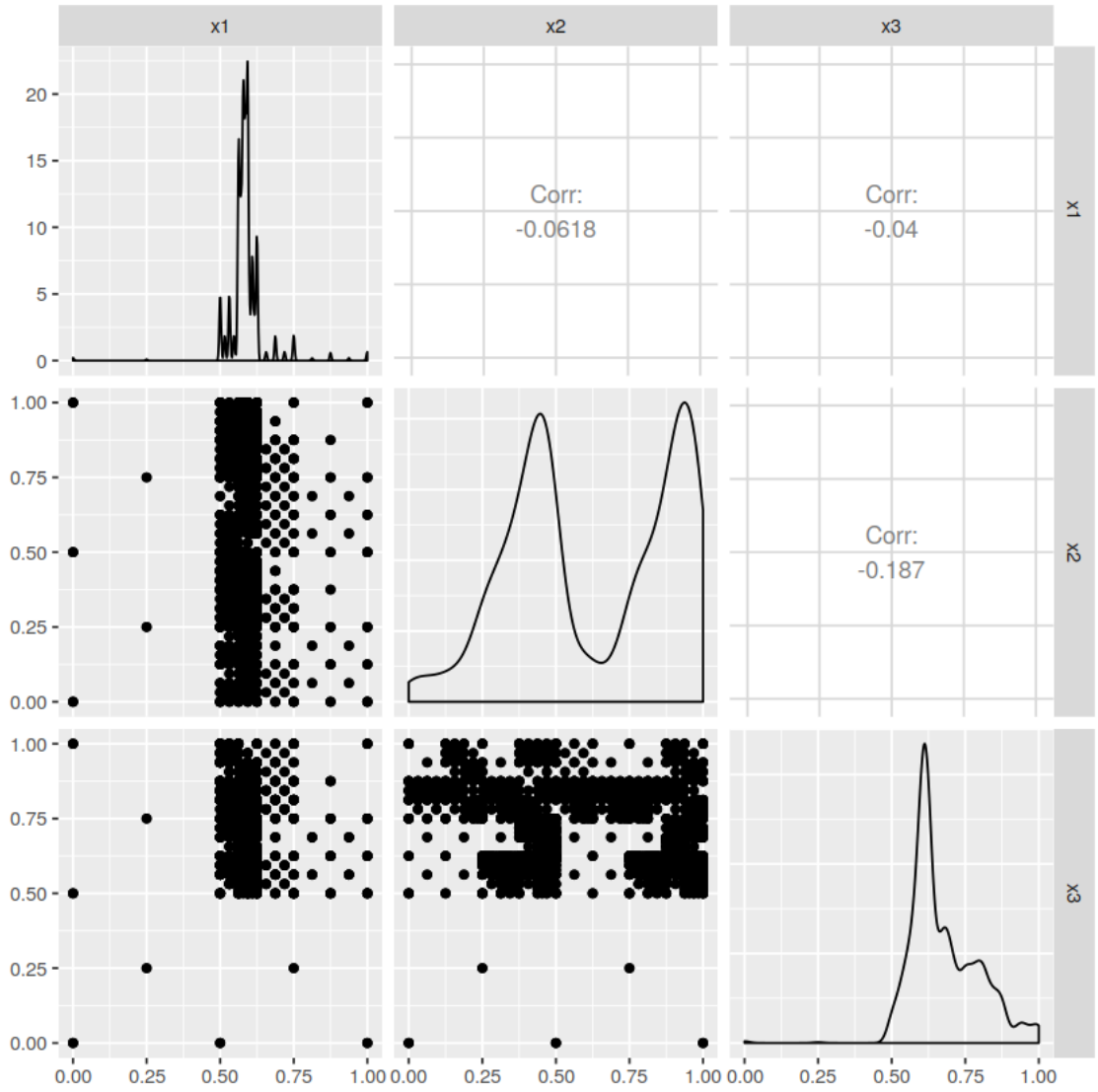
x3 vs x1.

[29]:
```
ggplot(
    xs[compute == FALSE, .(N=.N, Depth=max(mapply(ssa.depth, i1))), by=.(x3,␣
 ↪x1)],
    aes(x=x3, y=x1, size=N, color=Depth)
) + geom_point()
```



Pairs.

[30]:
```
ggpairs(xs, 6:8)
```


x1
x2
x3
Corr:
-0.0618
Corr:
-0.04
Corr:
-0.187
20
15
10
5
0
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00