Gmail **Marius Hofert <marius.hofert@gmail.com>**

Thesis - Grade

8 messages

Martin Štefánik <stefanik.mar@gmail.com> Sun, Dec 18, 2016 at 4:13 AM
To: Marius Hofert <marius.hofert@uwaterloo.ca>

Dear professor Hofert,

I had the presentation of my master thesis on Thursday and I wanted to let you know that I received a 6 as you suggested to prof. Embrechts as he told me :) Regardless of what will be next for me, I wanted to thank you for the cooperation and the time you found for our Skype calls, especially since I know that you had quite a busy period.

I will get in touch later with the documented functions for qrmtools. Enjoy the rest of your vacation and merry Christmas!

Best,
Martin

Marius Hofert <marius.hofert@uwaterloo.ca> Sun, Dec 18, 2016 at 12:47 PM
To: Martin Štefánik <stefanik.mar@gmail.com>

On Sun, Dec 18, 2016 at 5:13 AM, Martin Štefánik <stefanik.mar@gmail.com> wrote:
> Dear professor Hofert,
>
> I had the presentation of my master thesis on Thursday and I wanted to let
> you know that I received a 6 as you suggested to prof. Embrechts as he told
> me :)

Hi,

Congrats! :-))))

> Regardless of what will be next for me, I wanted to thank you for the
> cooperation and the time you found for our Skype calls, especially since I
> know that you had quite a busy period.

Yeah, it was/is quite busy, but I always enjoy(ed) talking to you.

>
> I will get in touch later with the documented functions for qrmtools.

Great.

I'll look into it as of tonight/early tomorrow (due to the revision).

> Enjoy the rest of your vacation and merry Christmas!

Thanks, for you, too.

Many cheers,
Marius

>
> Best,
> Martin

--
Marius Hofert, Dr. rer. nat.
Assistant Professor
Department of Statistics and Actuarial Science
Faculty of Mathematics
University of Waterloo
200 University Avenue West, Waterloo, ON, N2L 3G1
+1-519-888-4567, ext. 31394 (office M3 4207)
http://math.uwaterloo.ca/~mhofert

Martin Štefánik <stefanik.mar@gmail.com> Tue, Jan 3, 2017 at 10:42 AM
To: Marius Hofert <marius.hofert@uwaterloo.ca>

Dear professor Hofert,

please find attached the documented versions of the rearrangement function for the ABRA - block_rearrange() - and the ABRA() function itself. It took me some time because I had busy holidays and on top of all that I got quite sick and am still recovering.

I did not know the precise form in which you wanted to include the functions into the qrmtools package (some small adjustments might be needed so feel free to make them in any way you like or write me an email describing what do you want me to do) so I documented both functions separately. I think that everything that will be needed is in there.

Since I do not have the source code for the package, I was not able to check whether the links to other functions in the documentation render as they should.

One more thing. I tried to be consistent with the way qrmtools is already documented and when I was copying some parts, I have noticed one small mistake in the documentation of ARA() and RA(). In what the function returns, it is written that *converged* is a bivariate vector, but it is a trivariate vector and includes one more value. However, I am still looking into the latest version available on CRAN, so it is possible that this has already been changed.

All the best in the new year!

Cheers,
Martin
[Quoted text hidden]

2 attachments

ABRA.R
8K

block_rearrange.R
6K

Marius Hofert <marius.hofert@uwaterloo.ca> Fri, Jan 6, 2017 at 11:19 PM
To: Martin Štefánik <stefanik.mar@gmail.com>

Hi Martin,

Thanks a lot. I fixed all (two) previous bugs we discussed (longer ago) in rearrange(). I'm currently working on block_rearrange(). I adapted it slightly to be more 'similar' to rearrange():

1) I used the more readable (but longer) 'n.lookback' for the number of lookback steps (formerly: M). This is now also an argument for rearrange() (with the same default) :-)

2) I put in 'best.ES' as another method (like rearrange() has). I assume this can be computed with block_rearrange(), too. What do you think?

3) I now also allow relative tolerances (in variance), so we also have

a 'tol.type' in block_rearrange() now.

4) Instead of returning 'iter' (number of iterations), I returned the computed optimal row sums after each block rearrangement. The length of this vector is simply 'iter' and thus we return more (meaningful) information. I used the same trick as in rearrange() (doubling the size of this vector if necessary -- avoiding copying).

I attached the version I currently have (see the last function; the very end). I haven't tested it yet, though. There are two more things I'm wondering about:

1) Can you think of any possible scenario in which allowing 'tol' to be NULL makes sense? We have this feature in rearrange() where it means that we rearrange until all last n.lookback-many rearrangements actually lead to no rearrangements (that is, until all n.lookback-many columns are oppositely ordered to each other). What could this mean for block rearrangements? In rearrange() we look at one column at a time and then just need to check whether it changes or not. In block_rearrange() we could look at whether the whole picked block isn't changed. Hmmm... not sure how meaningful that is if different blocks have different (random) number of columns... What do you think? If it doesn't make sense, we can just describe it in the code and not allow it here.

2) It would be nice to have a 'trace' feature for block_rearrange() as we have for rearrange(). We could indicate with vertical bars which of the columns belong to the randomly picked block...

... that's how far I am. Let me know what you think. I should be able to continue tomorrow afternoon (my time zone).

Cheers,
M
[Quoted text hidden]

VaR_ES_bounds.R
47K

Martin Štefánik <stefanik.mar@gmail.com> Sat, Jan 7, 2017 at 4:55 AM
To: Marius Hofert <marius.hofert@uwaterloo.ca>

Dear professor Hofert,

several notes from my side:

1) Makes sense.

2) I cannot see the ES_np() function, but I do not see a reason why it should fail.

3) I skipped the relative convergence for block_rearrange intentionally. We are using variance instead of the minimal row sum, therefore we know that the target is 0 regardless of the margins - I think that absolute tolerance is preferable here. Additionally, having default tol = 0 will result in errors as the denominator in the relative convergence will be 0 upon termination. The very least I would do is to switch the order of tol.type to c("absolute", "relative").

4) I did not see that you have done that for rearrange after we discussed the concatenation approach being slower, but it makes sense to have it consistently.

As for the tol = NULL case, does it happen for rearrange() that it is more desirable than tol = 0? I do not see it being as sensible as in rearrange() for two reasons. First, it will be more time-consuming to check the be checking the whole block every time. Second, it is satisfactory to have n.lookback set to a low number when N is large, making the termination more likely to happen prematurely. Say if d=100 and N = 1024, then good n.lookback might be 5, while when you designed that option for rearrange() you would have essentially n.lookback = 100. What do you think?

Regarding tracing, that depends mostly on you I would say. It is your package after all. However, I think that given we have that for rearrange(), it would make sense, and looking at the tracing in rearrange(), it should not be too many additional lines of code. Should I do that?

One thing that comes into my mind is that we need to describe which block of columns is the one that is being changed. When tracing rearrange(), it is clear that we are changing a single column so the one that has a different colname from the rest is the one that is supposed the change. In block_rearrange, it can be either of the blocks, so if one block has colnames "=" and the other "|" we should say the block with "|" is changing (even though it is kind of intuitive from "=").

Cheers,
Martin

[Quoted text hidden]

Marius Hofert <marius.hofert@uwaterloo.ca> Sun, Jan 8, 2017 at 12:21 AM
To: Martin Štefánik <stefanik.mar@gmail.com>

On Sat, Jan 7, 2017 at 4:55 AM, Martin Štefánik <stefanik.mar@gmail.com> wrote:
> Dear professor Hofert,
>
> several notes from my side:
>
> 1) Makes sense.
>
> 2) I cannot see the ES_np() function, but I do not see a reason why it
> should fail.

Hi,

thanks for your quick reply.

ES_np() is the non-parametric expected shortfall estimator, it's in
the R-Forge version of 'qrmtools' (in a different file than the one I
sent).

>
> 3) I skipped the relative convergence for block_rearrange intentionally. We
> are using variance instead of the minimal row sum, therefore we know that
> the target is 0 regardless of the margins - I think that absolute tolerance
> is preferable here. Additionally, having default tol = 0 will result in
> errors as the denominator in the relative convergence will be 0 upon
> termination. The very least I would do is to switch the order of tol.type to
> c("absolute", "relative").

I saw that your docu used both: absolute tolerance for individual
convergence and relative tolerance for joint convergence. The latter
makes sense. My hope would be that one can also use relative tolerance
for the individual convergence (easiest, also to describe ... it's
getting messy with all those different functions...), but I'll have to
look into that again tomorrow (did this ever happen to you that the
row sums jump to 0 variance at some point?). Mixing the two types of
tolerances is not so 'nice'.

>
> 4) I did not see that you have done that for rearrange after we discussed
> the concatenation approach being slower, but it makes sense to have it
> consistently.

I think I always returned 'opt.row.sums' and this is now also what
block_rearrange() returns. You correctly pointed out early on that
c()-ing a single number to a vector (after every column rearrangement)
is time-consuming -- so I fixed that: we start with vectors of length
64 and double their size if required.

>
> As for the tol = NULL case, does it happen for rearrange() that it is more
> desirable than tol = 0?

I'd say rather 'no', but the original formulation of the RA said so
somewhere (rearrange until *each* column is oppositely ordered to the
sum of all others), that's why we implemented it.

> I do not see it being as sensible as in rearrange()
> for two reasons. First, it will be more time-consuming to check the be
> checking the whole block every time.

true. ... and we also know from our paper that one actually rarely
needs 'tol = NULL'. So I'll omit that from block_rearrange(). It would
make most sense if tol = NULL means "all possible blocks are
oppositely ordered to the sum of all others", which is impossible to
check, time-wise. Or at least, whether all columns are oppositely
ordered to the sum of all others -- but checking that is also too
time-consuming.

> Second, it is satisfactory to have
> n.lookback set to a low number when N is large, making the termination more
> likely to happen prematurely. Say if d=100 and N = 1024, then good
> n.lookback might be 5, while when you designed that option for rearrange()
> you would have essentially n.lookback = 100. What do you think?

that's true. But remember that rearrange() always *at least*
rearranged each column once, before even looking back. When I tested
this early one, I just compared two adjacent columns and determined
'convergence' based on that.... but the results I got was far off.

However, I had to rethink this a bit when introducing 'n.lookback' to
rearrange(). I know do not have this requirement of rearranging each
columns at least once anymore. But the default 'n.lookback' still
reflects that by choosing ncol(X) (as you had, too). So if a user
wants, he can determine convergence earlier or later.

> Regarding tracing, that depends mostly on you I would say. It is your
> package after all. However, I think that given we have that for rearrange(),
> it would make sense, and looking at the tracing in rearrange(), it should
> not be too many additional lines of code. Should I do that?

I put that in now. I saw that you actually rearrange the 'complement
block', not the randomly chosen 'block'. Of course that doesn't
matter, but I found it more readable to rearrange the block, so that's
in the new version as well.

Here are some more comments.

1) The BRA actually rearranges all columns in the randomly chosen
block in the same way, right? Almost seems a bit strange: Why not
pick a block and then for each column inside the block, rearrange it
differently (maybe itself with the RA? would probably be more optimal
but then also more time consuming).

2) I saw that you update the row sum after each block rearrangement by
computing the row sum over the whole matrix. Since we already have the
row sum over the 'complement block', we only have to update that row
sum with the row sums over all columns in the rearranged 'block'. That
should be faster (remember that we use the same trick in rearrange()),
at least if the randomly chosen block has a small number of columns.
Note: I already put that into the code.

3) The most important line is block_rearrange() is "X[, block] <-
X[order(rs.block),][indices_opp_ordered_to(rs.complement.block),

block]" where we oppositely reorder the block according to the complement block (I used the same helper function here as in rearrange()). The main difference to rearrange() is that that the operation of sorting X (so essentially 'X[order(rs.block),]') is only done once (see X.lst.sorted). I guess this is one reason why block_rearrange() is slower. As far as I can see, there's no hope for speeding this up, as the row sum over the picked blocks can change all the time... However, I could speed it up a little but as the above code aways sorts the *whole* matrix X yet we only need the part corresponding to the 'block'. This is now fixed and separated in two lines (to make it more readable).

Another reason why block_rearrange() is slower is that rearrange() works with lists, not matrices. I haven't tackled this yet, I guess due to the sorting all the time, the decrease in run time is probably not that large -- but at least we should mention it (to make the comparison fair). I now mention it at least in the code.

4) Does an 'is.sorted' option make sense for block_rearrange()?

Note: I just committed (see R-Forge). I'll fix the documentation tomorrow morning then.

Cheers,
M
[Quoted text hidden]

Martin Štefánik <stefanik.mar@gmail.com> Sun, Jan 8, 2017 at 5:41 AM
To: Marius Hofert <marius.hofert@uwaterloo.ca>

Hi,

The row sum variance is very rarely 0 itself, but it happened to me. Suddenly I was getting an error and had to go though the function in a debugging mode because I was not getting an error before that, and I found out that the problem was precisely that the row sum variance was 0. As you suggested by the question, I do not think that this should happen frequently (especially for larger N), but as it already happened to me, it might.

In case you want to have the relative convergence there, we might consider treating the case when the row sum variance actually reaches 0 to avoid errors completely. It is not very "clean", but the only thing that currently comes to my mind.

I have now thought about tol=NULL and I was not completely right. Even though having it will not guarantee that all blocks (row sums of the blocks) will be oppositely ordered, we do not necessarily have to wait n.lookback iterations for no rearrangement, but we could wait d iterations. This would solve the issue when n.lookback is low compared to d. Additionally, we do not have to check changes in the whole block, because the same permutation is applied to each column in the block. It is thus satisfactory to always check for a change in the first column of the block.

1) The idea behind the BRA is achieving Sigma-countermonotonicity https://arxiv.org/abs/1502.02130 - they basically say BRA is better at minimizing a dependence measure that they do not really refer to by any name, but could be described as multivariate rank correlation coefficient. Also, the RA is essentially a special case of this, where one block has always one column only and we always rearrange that particular block. (In the BRA it might happen that we have exactly the same situation, but we rearrange the larger block). What comes to my mind now is whether it would be worthwhile to examine whether always rearranging the block that is smaller would not be better. What do you think?

2) Agreed. I just probably missed that when I wrote the function, and later concentrated on stuff related to the lookback period and other.

3) Yes. I think we discussed this over the Skype and I definitely talk about this in the thesis as well. I do not think there is a way around this since essentially every time we are doing the opposite ordering it might happen that both vectors differ in at least one value from all the ones that we have rearranged before as well as from the original columns from the matrix of quantiles.

I have also noticed that I sort the whole matrix, but when I tried to change that in an easy way, I was getting errors. I thought that since I only use order() once (on the row sum) it should not be such a big difference and then let it go for a while.

As for the lists vs. matrices, I think that it is more complicated to manipulate lists in block_rearrange(), which is why I skipped it - in rearrange() we always manage to take advantage that we have one separated column, so it makes a lot of sense to have a list of single columns. I think that the reduction in runtime from selecting the columns quickly will be lost by having to manipulate it differently.

For instance, if we denote X.list the list of columns of X wouldn't:

```
X[order(rs.complement.block), ]
```

be faster than:

```
o <- order(rs.complement.block)
sapply(X.list, function(x) x[o])
```

?

Also .rowSums operates on matrices so in order to carry out the row sums efficiently, we would have to bind the randomly selected elements of the list together, compute the row sums and split them again. I simply though this cannot be more efficient than working with the matrices in this case.

4) I think it does not. Because of 3) we cannot exploit any previously sorted columns except for the first iteration (in which we know that row sum of any randomly selected columns will be sorted).

Best,
Martin
[Quoted text hidden]

Marius Hofert <marius.hofert@uwaterloo.ca> Sun, Jan 8, 2017 at 12:17 PM
To: Martin Štefánik <stefanik.mar@gmail.com>

On Sun, Jan 8, 2017 at 5:41 AM, Martin Štefánik <stefanik.mar@gmail.com> wrote:
> Hi,
>
> The row sum variance is very rarely 0 itself, but it happened to me.

... okay, I switched back to *absolute* individual tolerances and *relative* joint tolerances and also documented this now (and used a different argument name).

> Suddenly I was getting an error and had to go though the function in a
> debugging mode because I was not getting an error before that, and I found
> out that the problem was precisely that the row sum variance was 0. As you
> suggested by the question, I do not think that this should happen frequently
> (especially for larger N), but as it already happened to me, it might.
>
> In case you want to have the relative convergence there, we might consider
> treating the case when the row sum variance actually reaches 0 to avoid
> errors completely. It is not very "clean", but the only thing that currently
> comes to my mind.
>
> I have now thought about tol=NULL and I was not completely right. Even
> though having it will not guarantee that all blocks (row sums of the blocks)
> will be oppositely ordered,

... but that was my *definition* of tol = NULL. We have an 'operation' (rearrange: rearranging a single column; block_rearrange: rearrange a whole block) and tol = NULL should stand for 'all possible operations leading to no change'. As such we would need to check that all possible blocks did not change -- which we can't, for two reasons: 1) unlike with the column rearrangements, we don't cycle through all possible blocks, we randomly pick them; and 2) even if we would, there would simply be too many....

I'm not sure I can follow you, but let me know in case I'm missing a/the point.

> we do not necessarily have to wait n.lookback
> iterations for no rearrangement, but we could wait d iterations. This would
> solve the issue when n.lookback is low compared to d. Additionally, we do
> not have to check changes in the whole block, because the same permutation
> is applied to each column in the block. It is thus satisfactory to always
> check for a change in the first column of the block.
>
> 1) The idea behind the BRA is achieving Sigma-countermonotonicity
> https://arxiv.org/abs/1502.02130 - they basically say BRA is better at
> minimizing a dependence measure that they do not really refer to by any
> name, but could be described as multivariate rank correlation coefficient.
> Also, the RA is essentially a special case of this, where one block has
> always one column only and we always rearrange that particular block. (In
> the BRA it might happen that we have exactly the same situation, but we
> rearrange the larger block). What comes to my mind now is whether it would
> be worthwhile to examine whether always rearranging the block that is
> smaller would not be better. What do you think?

I'm sure this would work, too, but the bigger question is: In what
ways should columns be picked? Picking single ones at a time is one
option, picking blocks at random another. There are possibly many
others.

Given our computational advantages when looking at blocks of length 1
(= single columns), one could actually combine the two, staying with
'picking single columns at a time', but then picking them in a more
meaningful way than simply cycling through all columns. I could
potentially simply pick them at random, too. Ideally, we would have a
2nd operation, which is much faster than the reordering, which we
could apply to each of the columns and *then* decide which one we
reorder next. That would be cool. Or have that faster operation and
apply it to all possible blocks of size 2 (so d*(d-1)/2) and then
decide which block is rearranged next.

Nice idea to mention in the outlook of the paper, I'd say. Not sure
how practical it will be (overhead...)

>
> 2) Agreed. I just probably missed that when I wrote the function, and later
> concentrated on stuff related to the lookback period and other.
>
> 3) Yes. I think we discussed this over the Skype and I definitely talk about
> this in the thesis as well. I do not think there is a way around this

... ok (couldn't immediately see one either)

> since
> essentially every time we are doing the opposite ordering it might happen
> that both vectors differ in at least one value from all the ones that we
> have rearranged before as well as from the original columns from the matrix
> of quantiles.
>
> I have also noticed that I sort the whole matrix, but when I tried to change
> that in an easy way, I was getting errors. I thought that since I only use
> order() once (on the row sum) it should not be such a big difference and
> then let it go for a while.

indeed.

>
> As for the lists vs. matrices, I think that it is more complicated to
> manipulate lists in block_rearrange(), which is why I skipped it - in
> rearrange() we always manage to take advantage that we have one separated

> column,

true

> so it makes a lot of sense to have a list of single columns. I think
> that the reduction in runtime from selecting the columns quickly will be
> lost by having to manipulate it differently.
>
> For instance, if we denote X.list the list of columns of X wouldn't:
>
> X[order(rs.complement.block),]
>
> be faster than:
>
> o <- order(rs.complement.block)
> sapply(X.list, function(x) x[o])
>
> ?

I was very surprised when Kurt showed me the differences. Indeed, if
you work with vapply (or at least unlist(lapply())), working with
lists instead of subsetting matrices ([,]) is quite a bit faster.
Essentially, in a matrix you first have to find the starting point and
end point representing that column you pick out. In a list you can
directly access it.

>
> Also .rowSums operates on matrices so in order to carry out the row sums
> efficiently, we would have to bind the randomly selected elements of the
> list together, compute the row sums and split them again. I simply though
> this cannot be more efficient than working with the matrices in this case.

In rearrange() we can avoid the call to rowSums() completely, as we
have the row sums before the rearrange and we update that by the
change in the current column.

This would of course still work if a randomly chosen block is of size
one, but once it's larger, one cannot really avoid the call of
rowSums() anymore.
So I'd leave it the way it is -- but if we compare run times in the
paper, we have to mention this drawback since a referee could complain
about an 'unfair' comparison.

>
> 4) I think it does not. Because of 3) we cannot exploit any previously
> sorted columns except for the first iteration (in which we know that row sum
> of any randomly selected columns will be sorted).

ok, thanks.

I now finished all the documentation and committed. Feel free to call
block_rearrange() and ABRA() in all sorts of ways to 'test' them.
Install from R-forge via

```
install.packages("qrmtools", repos = "http://R-Forge.R-project.org")
```

For the paper: It would be good if the results we present there could
all be reproduced (including graphics), maybe with a smaller sample
size for those that run too long (or we keep them separate). As such,
if you have a stand-alone R script for those results then, we can put
it as a demo in the package and refer to it from within the paper.
Reviewers like that :-) [... or so I hope :-)]

I gotta go back to the lecture preparation now...

By the way, any 'news' concerning the 'future'?

Cheers,
M
[Quoted text hidden]