
Subject: Re: kronecker product

Posted by [Yngvar Larsen](#) on Tue, 06 Dec 2011 14:31:40 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Dec 6, 2:18 am, Ecaterina Coman <ecatc...@umbc.edu> wrote:

> Hey guys,
> I have two sparse arrays in IDL that I want to compute the kronecker
> product with. I've googled like a crazy person but can't find anything
> useful. Does IDL had any routines that I can use to get the krocker
> product of these two sparse arrays?
> Thanks.

This straightforward implementation should work. It is probably possible to optimize it.

```
function kroneckerProduct, A, B
;+
;
; Kronecker tensor product of A and B.
; Result is a large matrix formed by taking all
; possible products between the elements of A and those of B.
;
; Input:  A - (a x b) matrix
;         B - (c x d) matrix
;
; Output: C - (ac x bd) matrix
;
; C =
;   A(0 , 0) B   A( 0, 1) B   ... A( 0, n-1) B
;   A(1 , 0) B   A( 1, 1) B   ... A( 1, n-1) B
;   :           :           :
;   A(m-1, 0) B   A(m-1, 1) B   ... A(m-1, n-1) B
;
; Written by Yngvar Larsen <yngvar.larsen@norut.no>
; 2004-07-01
;
; Fixed type bug. Let IDL do the type inference:
; Now returns array of type 'size(A[0]*B[0], /type)'.
; YL 2005-01-13
;-

sizeA = size(A, /dimensions)
sizeB = size(B, /dimensions)
if n_elements(sizeA) eq 1 then sizeA = [sizeA, 1]
if n_elements(sizeB) eq 1 then sizeB = [sizeB, 1]

C = make_array(sizeA[0]*sizeB[0], sizeA[1]*sizeB[1], $
               type=size(A[0]*B[0], /type))
```

```
for x=0L,sizeA[0]-1 do begin
  ix = x*sizeB[0]
  for y=0L,sizeA[1]-1 do C[ix,y*sizeB[1]] = A[x,y]*B
endfor
```

```
return, C
end
```

```
--
Yngvar
```

Subject: Re: kronecker product
Posted by [Oana Coman](#) on Thu, 08 Dec 2011 02:05:15 GMT
[View Forum Message](#) <> [Reply to Message](#)

I tried the code, but am getting an "Struct expression not allowed in this context: <STRUCT Array[1]>."
I'm thinking it is because that particular code does not work with sparse matrices?

Subject: Re: kronecker product
Posted by [Yngvar Larsen](#) on Sat, 10 Dec 2011 16:26:29 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Dec 8, 3:05 am, Oana Coman <ecatc...@gmail.com> wrote:
> I tried the code, but am getting an "Struct expression not allowed in
> this context: <STRUCT Array[1]>."
> I'm thinking it is because that particular code does not work with
> sparse matrices?

Sorry. I interpreted your question wrong. One obvious workaround is to expand the matrices first, use my function and then transform the answer to a sparse representation:

```
Cfull = kroneckerProduct(fulstr(Asparse), fulstr(Bsparse))
C = sprsin(Cfull)
```

But then then point of using sparse matrices is gone. Also, CFULL might be very large.

Here is an implementation that should work for large sparse matrices:
8<-----

```
function kronecker_sparse, A, B, double=double, thresh=thresh
```

```

if keyword_set(double) $
then type = 5      $
else type = size(A.sa[0]*B.sa[0], /type)

if keyword_set(thresh) $
then threshold = thresh $
else threshold = (machar(double=(type eq 5))).eps

Na = A.ija[0] - 2      ; A is (Na x Na)
Nb = B.ija[0] - 2      ; B is (Nb x Nb)
Nc = Na*Nb

Noda = n_elements(A.ija)-1-Na      ; # off-diagonal elements in A
Nodb = n_elements(B.ija)-1-Nb      ; # off-diagonal elements in B
Nda = total(A.sa[0:Na-1] gt threshold) ; # nonzero diagonal elements
in A
Ndb = total(B.sa[0:Nb-1] gt threshold) ; # nonzero diagonal elements
in B

;; clen = (# diagonal elements in C) + 1 + (# nonzero offdiagonal
elements in C)
Nnzc = (Noda+Nda)*(Nodb+Ndb) ; total # nonzero elements in C
Nnzdc = Nda*Ndb      ; # nonzero diagonal elements in C
Nodc = Nnzc - Nnzdc      ; # nonzero offdiagonal elements in C
clen = Nc + 1 + Nodc

C = {sa: make_array(clen, type=type), $
    ija: lon64arr(clen)}

C.sa[0:Nc-1] = B.sa[0:Nb-1]#A.sa[0:Na-1] ; Diagonal elements
C.ija[0] = Nc + 2
if (Nodc gt 0) then begin
    bCache = ptrarr(Nb)
    kk = 0L
    for ii=0L, Na-1 do begin
        ;; Find nonzero elements of row #ii of A
        if (A.ija[ii+1] gt A.ija[ii]) then begin
            offElem = A.ija[A.ija[ii]-1:A.ija[ii+1]-2]-1
            aind = [ii, offElem]
            aVal = [A.sa[ii], A.sa[A.ija[ii]-1:A.ija[ii+1]-2]]
        endif else begin
            aind = [ii]
            aVal = [A.sa[ii]]
        endelse
        nzinda = where(abs(aVal) gt threshold, nzaii)
        if (nzaii eq 0) then begin
            ;; No nonzero elements in row #ii of A
            ;; Insert Nb empty rows in C.

```

```

C.ija[kk+1:kk+Nb] = C.ija[kk]
kk += Nb
endif else begin
;; Sort nonzero elements in row #ii of A
aInd = aInd[nzinda]
aVal = aVal[nzinda]
sort = sort(aInd)
aInd = aInd[sort]
aVal = aVal[sort]
for jj=0, Nb-1 do begin
  if (ii eq 0) then begin
    cache = {N: 0L, bind: ptr_new(), bVal: ptr_new()}
    ;; Find nonzero elements of row #jj of B
    if (B.ija[jj+1] gt B.ija[jj]) then begin
      offElem = B.ija[B.ija[jj]-1:B.ija[jj+1]-2]-1
      bind = [jj, offElem]
      bVal = [B.sa[jj], B.sa[B.ija[jj]-1:B.ija[jj+1]-2]]
    endif else begin
      bind = [jj]
      bVal = [B.sa[jj]]
    endelse
    nzIndb = where(abs(bVal) gt threshold, nzbjj)
    cache.N = nzbjj
    if (nzbjj gt 0) then begin
      ;; Sort nonzero elements in row #jj of B
      bind = bind[nzIndb]
      bVal = bVal[nzIndb]
      sort = sort(bind)
      bind = bind[sort]
      bVal = bVal[sort]
      cache.bInd = ptr_new(bind)
      cache.bVal = ptr_new(bVal)
    endif
    bcache[jj] = ptr_new(cache)
  endif else begin
    nzbjj = (*(bcache[jj])).N
    if (nzbjj gt 0) then begin
      bInd = (*(bcache[jj])).bInd
      bVal = (*(bcache[jj])).bVal
    endif
  endelse
  if (nzbjj eq 0) then begin
    ;; No nonzero elements in row #jj of B.
    ;; Insert an empty row in C.
    C.ija[kk+1] = C.ija[kk]
  endif else begin
    cVal = bVal#aVal
    cInd = lonarr(nzaai*nzbjj)
  end
end
end

```

```

        for ll=0, nzaii-1 do clnd[ll*nzbjj] = aind[ll]*Nb +
bind
        ;; Remove diagonal element of C which is already
stored.
        ind = where(clnd ne kk, count)
        if (count eq 0) then begin
            ;; No nonzero offdiagonal elements in row #jj of
B.
            ;; Insert an empty row in C.
            C.ija[kk+1] = C.ija[kk]
        endif else begin
            clnd = clnd[ind]
            cVal = cVal[ind]
            nzindc = where(abs(cVal) gt threshold, nzc_row)
            if (nzc_row eq 0) then begin
                ;; No nonzero offdiagonal elements in row #jj
of B.
                ;; Insert an empty row in C.
                C.ija[kk+1] = C.ija[kk]
            endif else begin
                ;; Insert offdiagonal elements in row #kk of C.
                C.ija[kk+1] = C.ija[kk] + nzc_row
                C.ija[C.ija[kk]-1:C.ija[kk]+nzc_row-2] =
clnd[nzindc]+1
                C.sa[C.ija[kk]-1:C.ija[kk]+nzc_row-2] =
cVal[nzindc]
            endelse
        endelse
        kk++
    endfor
endelse
endfor
endif else C.ija[0:clen-1] = Nc + 2
heap_free, bCache

return, C
end
8<-----

--
Yngvar

```
