Deni Vale[1], Nils Paar[2]

[1]Istrian University of Applied Sciences, Riva 6, 52100 Pula, Croatia
[2]University of Zagreb, Faculty of Science, Department of Physics, Bijenička cesta 32, 10000 Zagreb, Croatia

Email: dvale@iv.hr, npaar@phy.hr

Pseudocode description of algorithm for fast calculation of full Wick's contractions in quantum many-body fermion systems (FWC-QMBFS)

Created: April 5 2020
Last modification: May 11 2023

Published with corresponding code in C programming language on May 11 2023

Algorithm 1. Pseudocode for fast calculation of full Wick's contractions with canonical transformation for fermionic quantum mechanical many-body systems.

Function contract_one_line()

Input: two operators a and b

Output: modified Kronecker delta δ, logical *exist*

$\delta \leftarrow$ empty; *exist* $\leftarrow$ false

if operator type a **is not equal** operator type b **return**

if operator a **is** of annihilation **and** b **is** of creation type **then**

Store quantum numbers of a and b in δ Store type of a (or b) in δ

exist $\leftarrow$ true

endif

return

end function

Function contract_wick_lines()

Input: points on circle or regular polygon T, directed lines L, all operators O, order of appearance *ord*

Output: contraction result *output*

output $\leftarrow$ empty

for i = 1 to n/2 **do**

($exist$, δ) $\leftarrow$ contract_one_line($O(L_i(beg))$, $O(L_i(end))$)

if *exist* = *true* **return**

Store modified Kronecker delta δ in *output*

end for

$I \leftarrow$ empty

if $n/2 > 1$ **then**

(I, k) $\leftarrow$ calc_intersec(T, L, O)

else

$k \leftarrow 0$

endif

Phase of Wick's contraction $\varphi = (-1)^k$

Store directed lines L, intersection points I, phase φ, order of appearance *ord* in *output*

return

end function

Function calc_intersec()

Input: points on circle or regular polygon T, directed lines D, all operators O

Output: intersection points I

Lines $L \leftarrow$ empty

Step 1: Get line equations in explicit form:

for i = 1 to n/2 **do**

  $L_i \leftarrow$ get_coeff_of_line($T(D_i(beg))$, $T(D_i(end))$)

end for

Step 2: Calculate intersection points if exist:

$k \leftarrow 0$; $I \leftarrow$ empty

for i = 1 to n/2 **do**

  for j = 1 to n/2 **do**

   $J \leftarrow$ check_intersections(L_i, L_j)

   if I_k ! = empty **then**

    $I_k \leftarrow J$

    $k \leftarrow k + 1$

   endif

  end for

end for

return

end function

Function check_intersections()

Input: lines L_1 and L_2,

Output: intersection points I, logical n_c

$I \leftarrow$ empty; $n_c \leftarrow$ false

$D_S \leftarrow A(L_1)B(L_2) - A(L_2)B(L_1)$

if $|D_S|$ < eps **return**

 $D_X \leftarrow C(L_1)B(L_2) - B(L_1)C(L_2)$

 $D_Y \leftarrow A(L_1)B(L_2) - C(L_1)A(L_2)$

 x $\leftarrow D_X / D_S$

 y $\leftarrow D_Y / D_S$

if sqrt($x(I)^2 + y(I)^2$) ≤ 1 **then**

  $I \leftarrow$ empty

  $n_c \leftarrow$ true

endif

return

end function

Function get_coeff_of_line()

Input: points T_1 and T_2

Output: line L

$A \leftarrow y(T_1) - y(T_2)$

$B \leftarrow x(T_2) - x(T_1)$

$C \leftarrow y(T_1)B + Ax(T_1)$

```
        Store coefficients A, B and C in line L
        return
end function

Function pair_test()
        Input: operators O, total number of operators n
        Output: number of particle p and hole pairs h, logical exists
        p ← 0; h ← 0; exists ← false
        if n mod 2 = 1 return
        for i = 1 to n
                if type of O_i is hole operator then
                        if O_i is of creation type then
                                h_c ← h_c + 1
                        else
                                h_a ← h_a + 1
                        if h_c > h_a return
                endif
                else if O_i is of creation type then
                                p_c ← p_c + 1
                        else
                                p_a ← p_a + 1
                        endif
                        if p_c > p_a return
                        endif
        end for
        if p_c ! = p_a or h_c ! = h_a return
        p ← p_c; h ← h_c
        exists ← true
        return
end function

Function get_all_permutations()
        Input: indices l and r, total number of positions n
        Output: all permutations of positions (2D array) P
        Inout: positions of operators of the same type (1D array) I
        static k ← 0
        if l = r then
                Store particular permutation P_k ← I
                k ← k + 1
        else
                for i = l to r do
                        Swap I_l and I_i
```

```
            call get_all_permutations(I, l + 1, r, n, P)
            Swap I_l and I_i
        end for
    endif
end function

Function capm1comb()
    Input: array of changes of general operators F, number of general operators n
    Output: all possible transformations of general operators Q (2D array)
    Q ← empty
    p ← 2^n
    if n = 0 return
    for k = 1 to n
        l ← 2^k
        i ← 1
        B ← empty
        for j = 1 to l do
            B_j ← i
            i ← i + p/l
        end for
        i← 1; c ← −1
        for i = 1 to p do
            if i = B_j then
                c ← c · (−1)
                j ← j + 1
            endif
            if F_k = 1 then
                Q_k,i ← c
            else
                Q_k,i ← −c
            endif
        end for
    end for
    return
end function

Function prepair_combinations()
    Input: all operators O
    Output: all combinations of operators (2D array) Q
    Step 1. Look for appearance of operators of general type. Prepare all possible
    alternations of −1 and 1 and store them in 2D array:
    Q ← empty
```

Determine type of change for general operators and store it in array F
$G \leftarrow$ capm1comb(F, n)
$k \leftarrow 0$

Step 2. Transform any general quantum mechanical operator of general into particular type (canonical transformation to particle-hole many-body picture):

```
for i = 1 to n do
        for j = 1 to N_C do
        Q_{j,i} ← O_i
         if  O_i  is general type operator then
         for j = 1 to N_C do
                c ← G_{k,j}
                if F_k = 1 then
                        if c = 1 then
                                type(Q_{j,i}) ← particle creation
                        else
                                type(Q_{j,i}) ← hole annihilation
                        end if
                else
                        if c = −1 then
                                type(Q_{j,i}) ← particle annihilation
                        else
                                type(Q_{j,i}) ← hole creation
                        end if
                endif
              end for
         endif
    end for
end for
return
end function
```

Function map_operator_geo_point()

Input: all operators O, number of operators n
Output: points on circle or regular polygon T

```
T ← empty
ϕ ← arbitrary set to some value [0,2π]
if n < 2 return
for i = 1 to n do
   x(T_i) ← cos(2iπ/n + ϕ)
   y(T_i) ← sin(2iπ/n + ϕ)
end for
```

return
end function

Function fwcoc()

Input: all operators O, order of appearance *ord*

Output: all contractions for particular combination C

Step 1: Test all vanishing cases:

$C \leftarrow$ empty

$(exists, n_p, n_h) \leftarrow$ pair_test(O)

if *exists* = false **return**

Step 2: Preparation of geometric interpretation of Wick's contraction for one combination:

$X_h \leftarrow$ empty; $X_p \leftarrow$ empty; $D \leftarrow$ empty; $T \leftarrow$ empty

Find quantum numbers in O of hole annihilation operators if exist and store them in X_h

Find quantum numbers in O of particle annihilation operators if exist and store them in X_p

Create beginning of directed line and store it in D

$T \leftarrow$ map_operator_geo_point(O)

Number of permutations for creation operators is given by $p_p \leftarrow c_p!$,

$p_h \leftarrow c_h!$

$P_h \leftarrow$ get all_permutations(X_h, n_h, p_h)

$P_p \leftarrow$ get all_permutations(X_p, n_p, p_p)

Step 3: Evaluate nonvanishing permutations of creation operators:

$i \leftarrow 0$

for $h = 1$ to p_h **do**
  for $l = 1$ to n_h **do**
  $D_l(end) \leftarrow X_h$
  for $p = 1$ to p_p **do**
    for $k = 1$ to n_p **do**
      $D_{k+n_h}(end) \leftarrow X_p$
      $(K, exists) \leftarrow$ **contract_wick_lines**(T, D, O)
      if *exists* = true **then**
        Store result of contraction in C as $C_i \leftarrow K$
        $i \leftarrow i + 1$ **endif**
        end for
    end for
  end for
end for
return
end function

Function calculate_all_comb_of_wick_contraction()

Input: expression inside brackets with calculation parameters *prep*

Output: logical *success*

Step 1. Calculate all possible transformations of general operators:

success ← false

Extraction of second quantization operators O ← *braket*(*prep*)

All Wick's contractions W ← empty

Number of possible combinations $N_C \leftarrow 2^{n_a}$

All possible operators Q ← prepair_combinations(O, N_C)

Step 2. Calculate Wick's contractions for all possible canonical transformations of general operators into particle-hole picture:

$i \leftarrow 0$

for $k = 1$ to N_C **do**

(*exists*, K) ← fwcoc (O, k)

if *exists* = true **then**

Write K to standard output

Store contraction result for particular nonvanishing canonical transformation in $W_i \leftarrow K$

$i \leftarrow i + 1$

end if

end for

Step 3. Output as latex or/and gnuplot file (only descriptive here):

if *latex option* = true

Prepare latex headers for latex output

Prepare initial operators for latex output

for $k = 1$ to N_C **do**

Prepare canonically transformed operators for latex output

Prepare Wick's contraction lines for latex output

Prepare Kronecker's deltas together with heavyside theta functions for latex output

end for

if *pdf output* = true **then**

Write all prepared objects in latex file

call external program pdflatex to create pdf output

end if

if *show latex output* = true **then**

call external (arbitrary) program to show pdf

end if

end if

if *gnuplot option* = true

for $k = 1$ to N_C **do**

Write drawing data, i.e. operator points *T*, intersection points I and directed lines *D* in separate files

Create gnuplot script

if *eps output* = true **then**

run gnuplot script with external program gnuplot

Write gnuplot output of particular contractions to eps file **endif**

if *show eps output* = true **then**

call external program to show eps

endif

end for

end if

Function framework()

Input: command line options *opt*

Output: logical *success*

Step 1. Initialize (descriptive only):

success ← false

Read calculation parameters and store them in *prep*

Read input expression inside of quantum mechanics brackets and store them in *prep*

if reading anything above failed **return**

Step 2. Check (descriptive only):

Initial check of all objects in expression

if check failed **return**

Transformation of operator type **if** predefined indices are used

Step 3. Calculate full Wick contractions:

success ← calculate all_comb_of_wick_contraction(*prep*)

return

end function

Function main()

success ← false

Check for command line arguments and store them in *opt*

switch based on *opt*:

case file or stdin:

success←framework(*opt*) **break**

case help:

Write help message

break

else cases:

Write error message for reading command line arguments

break

end switch

Write status based on *success* **return**

end function