```
#!/bin/bash
#
# This script runs Gnuplot for several EMPIRE and/or TALYS data files
# Gnuplot plots data for total cross section comparison with EXFOR, TENDL
# and experimental data files.
#
# Script name: data-plot
# Author: Alex Björkholm 2013-2016
# Date: 2016.08.16


################################ WORKING NOTES ################################

# NOTE: Always check with the EXFOR database for missing data. https://www-nds.iaea.org/exfor/exfor.htm
# NOTE: TENDL database not implemented




################################ DECLARATIONS ################################

echo -e "\n
 ---------------------------\n
      CROSS SECTION PLOT    \n
 ---------------------------\n"

IFS=$'\n'

declare -a data_plot
declare -a input_data_bases
declare -a data_bases
declare -a input_search

declare -a dir_empire
declare -a dir_talys
declare -a dir_tendl
declare -a dir_exper
declare -a dir_exper
declare -a dir_gnuplot

declare -a file_paths_empire
declare -a file_paths_talys
declare -a file_paths_tendl
declare -a file_paths_exfor
declare -a file_paths_exper

declare -a input_reaction
declare -a input_products
declare -a natural_isotopes

declare -a targets
declare -a targetsAA
declare -a targetsZZ
declare -a targetsNN
declare -a targetsXX
declare -a targetsPP
declare -a targetsMM
declare -a temp_targets
declare -a temp_targets_MM
declare -a temp_targets_PP

declare -a beams
declare -a beamsAA
declare -a beamsZZ
declare -a beamsNN
declare -a beamsXX
declare -a temp_beams

declare -a products
declare -a productsAA
declare -a productsZZ
declare -a productsNN
declare -a productsXX
declare -a productsMM
declare -a input_products
declare -a temp_products_MM
declare -a temp_products

declare -a reactions
declare -a temp_reactions
declare -a emissions

declare -a delete_array

cd $HOME




################################ VARIABLES ################################

# Paths $RESEARCHDIR, $RESEARCHPROGRAMDIR, $EMPIREDIR, $TALYSDIR must be defined in bash

# Directory for this program and its reference data
programdir="$RESEARCHPROGRAMDIR/Cross section - Data plot scripts - Bash Gnuplot"
reference="$programdir/data-plot-ref.dat"
```

```

# Directories where EMPIRE or TALYS calculations are stored. Mandatory format for subdirectories is directories starting with "target,beam" with subsequent tags separated with "-". For example "75Br,h", "208Pb,p-run", "209Bi,a-omp-rot", and "202Hg,10B-omp".
dir_empire+=($(find $RESEARCHDIR -maxdepth 2 -type d -iwholename $RESEARCHDIR\*Calculations\ Empire\*))
dir_talys+=($(find $RESEARCHDIR -maxdepth 2 -type d -iwholename $RESEARCHDIR\*Calculations\ Talys\*))

# Directories where EXFOR and TENDL data are stored.
# Exfor data (download from http://www-nds.iaea.org/x4toc4-master/) should be sorted with the parseC4.py script in Empire to achieve the correct structure.
# Tendl data (download from ftp://ftp.nrg.eu/pub/www/talys/tendl2014/tendl2014.html) should be stored in the structure provided from the Tendle database.
dir_tendl+=($(find $RESEARCHDIR -maxdepth 3 -type d -iwholename $RESEARCHDIR\*Experimental\ and\ theoretical\ data\*tendl\*))
dir_exfor="$EMPIREDIR/EXFOR"
#dir_tendl="$TALYSDIR/TENDL"

# Directories where additional data in tab separated format is stored. Mandatory format for subdirectories is directories starting with "target,beam" with subsequent tags separated with "-". For example "75Br,h", "209Bi,a-omp-rot", and "202Hg,10B-omp".
dir_exper+=($(find $RESEARCHDIR -maxdepth 3 -type d -iwholename $RESEARCHDIR\*Experimental\ and\ theoretical\ data\*exper\* -not -iwholename \*chisq\*))
dir_gnuplot+=($(find $RESEARCHDIR -maxdepth 3 -type d -iwholename $RESEARCHDIR\*Experimental\ and\ theoretical\ data\*gnuplot\*))

# Directories where output from plots are saved.
mkdir -p $HOME/gnuplot-plots
gnuplot_plot="$HOME/gnuplot-plots"

# Directory for temp files used by GNUPLOT.
mkdir -p $HOME/gnuplot-temp
gnuplot_temp="$HOME/gnuplot-temp"




################################## FUNCTIONS ###################################

manual () {
  echo -e "This script runs Gnuplot for several EMPIRE and/or TALYS data files.
     Program data-plot input string:

     1.  Cross section data to plot
         product options: xs, direct, preequilibrium, compound, all-xs
         production options: activity, \#isotopes, yield, production
         emission options: xs, yield, all-xs
     2.  System(s) to plot
         options: exfor, exper, gnuplot, tendl, talys, empire
         multiple options can be chosen by writing option1_option2_option3
     3.  Reaction input
         options: '75As(h,x)76Br-75Br', 'natCr(p,x)52Mn', '52Cr(p,x)52mMn', '52Cr(p,x)52gMn' etc.
         multiple options can be chosen by writing option1_option2_option3
         product note: set outgoing particles to 'x' (calculated automatically)
     4.  Search string for EMPIRE and TALYS calculations
         multiple options can be chosen by writing option1_option2_option3
         wildcard notation * can be used to get the optimal selection
     5.  Graphical terminal type
         options: edit/no-edit/eps/epslatex/svg/table
     6.  Optional x-axis range in the form '5-33'
         if no range is defined the x-axis will autoscale to data

     Example: data-plot xs talys_empire 'natCr(p,x)52Mn' 'optimal' no-edit 5-40
              data-plot xs exfor_exper '75As(h,x)76Br-75Br' 'calculation' edit 5-50
              data-plot xs exfor_exper '75As(h,x)all' 'omp*HFB' edit 5-50\n"
  exit 2 ; }


natural_conversion () {
  local input=$1
  unset AAA; unset NAA; unset NAP

  if [[ $input != *[0-9]* ]] && [[ $input == nat[A-Z]* ]]  ; then
    NZZ=$(echo $input | sed 's/nat//g')
    zZZ=$(awk -v src=$NZZ '$2 == src { print $1 }' $reference)
    ZZZ=($(echo ${zZZ} | sed 's/^0*//'))

    while IFS=' ' read -r -a line; do
      # Extracting isotope atomic numbers for natural target
      AAA+=($(echo ${line[1]} | sed 's/^0*//'))
      # Extracting isotope percentage for natural target
      PP+=($( bc -l <<< "scale=6;${line[2]} / 100" | sed 's/^\./0./'))
    done < $TALYSDIR/structure/abundance/z${zZZ}

  fi

  if [[ $AAA != *[0-9]* ]] || [[ $ZZZ != *[0-9]* ]]; then
    echo -e "ERROR: natural_conversion function for input '$input'\n\n"; exit 2;
  fi ; }


nuclide_conversion () {
  local input=$1

  # If format of the beam is n, p, h, a etc., resolving ZZZ and AAA
  if [[ $input != *[0-9]* ]]; then
    AAA=$(awk -v src=$input '$1 == src { print $5 }' $reference)
    NZZ=${input}
    zZZ=$(awk -v src=$input '$1 == src { print $3 }' $reference)
    if [[ $input != n ]]; then ZZZ=($(echo ${zZZ} | sed 's/^0*//')); fi
```

```
    if [[ $input == n ]]; then ZZZ=($(echo ${zZZ})); fi
  fi

  # If format of target is ZZZAAA (083209), resolving ZZZ and AAA
  if [[ $input != *[a-z]* ]]; then
    AAA=$(echo ${input:3:7} | sed 's/^0*//')
    ZZZ=$(echo ${input:0:3} | sed 's/^0*//')
    zZZ=$(echo ${input:0:3})
    NZZ=$(awk -v src=$zZZ '$1 == src { print $2 }' $reference)
  fi

  # If format of the nuclide  is AAAXX (209Bi), resolving ZZZ and AAA
  if [[ $input == *[0-9][A-Z]* ]]; then
    AAA=$(echo $input | sed 's/[^0-9]*//g')
    NZZ=$(echo $input | sed 's/^[0-9]*//g')
    zZZ=$(awk -v src=$(echo $NZZ | sed 's/^m//g') '$2 == src { print $1 }' $reference)
    ZZZ=($(echo ${zZZ} | sed 's/^0*//'))
  fi

  if [[ $AAA != *[0-9]* ]] || [[ $ZZZ != *[0-9]* ]]; then
    echo -e "ERROR: nuclide_conversion function for input '$input'\n\n"; exit 2
  fi
  NNN=$(($AAA - $ZZZ)); }


p-n-emission () {
  local particle_emission=$1
  emission_number=0 ; NOUT=0 ; POUT=0

  if [ $particle_emission == gamma ]; then particle_emission=g; else
  particle_emission=$(echo $particle_emission | grep -o '[0-9]*[a-z]')

    for each_emission in ${particle_emission[@]}; do
    emission_particle=$(echo $each_emission | grep -o '[a-z]')
    emission_number=$(echo $each_emission | grep -o '[0-9]*')

    if [ -z $emission_number ]; then emission_number=1; fi

    n=$(awk -v src=$emission_particle '$1 == src { print $4 }' $reference)
    p=$(awk -v src=$emission_particle '$1 == src { print $3 }' $reference)
    NOUT=$(($NOUT + $emission_number * $n))
    POUT=$(($POUT + $emission_number * $p))

    done

  fi; }


particle-emission () {
  NOUT=$(($TNN + $BNN - $PNN))
  POUT=$(($TZZ + $BZZ - $PZZ))

  particle_emission=${NOUT}n${POUT}p
  if [ $particle_emission == 1n1p ]; then particle_emission=d; fi
  if [ $particle_emission == 2n1p ]; then particle_emission=t; fi
  if [ $particle_emission == 1n0p ]; then particle_emission=n; fi
  if [ $particle_emission == 0n0p ]; then particle_emission=gamma; fi
  if [[ $particle_emission == *n0p ]]; then particle_emission=$(echo $particle_emission | sed 's/0p//g'); fi
  if [[ $particle_emission == *-* ]]; then particle_emission=none; fi; }


MT_number () {
  local MT=()
  declare -a MT

  # See endf-6 manual p. 306-316. Here simplified since we're only interested in the produced nuclei
  if [ $POUT == 0 -a $NOUT == 0 ];then MT+=(2); fi
  if [ $POUT == 0 -a $NOUT == 1 ];then MT+=(4); fi
  if [ $POUT == 1 -a $NOUT == 3 ];then MT+=(11); fi
  if [ $POUT == 0 -a $NOUT == 2 ];then MT+=(16); fi
  if [ $POUT == 0 -a $NOUT == 3 ];then MT+=(17); fi
  if [ $POUT == 2 -a $NOUT == 3 ];then MT+=(22); fi
  if [ $POUT == 6 -a $NOUT == 7 ];then MT+=(23); fi
  if [ $POUT == 2 -a $NOUT == 4 ];then MT+=(24); fi
  if [ $POUT == 2 -a $NOUT == 5 ];then MT+=(25); fi
  if [ $POUT == 1 -a $NOUT == 1 ];then MT+=(28); fi
  if [ $POUT == 4 -a $NOUT == 5 ];then MT+=(29); fi
  if [ $POUT == 4 -a $NOUT == 6 ];then MT+=(30); fi
  if [ $POUT == 1 -a $NOUT == 2 ];then MT+=(32); fi
  if [ $POUT == 1 -a $NOUT == 3 ];then MT+=(33); fi
  if [ $POUT == 2 -a $NOUT == 2 ];then MT+=(34); fi
  if [ $POUT == 3 -a $NOUT == 4 ];then MT+=(35); fi
  if [ $POUT == 5 -a $NOUT == 7 ];then MT+=(36); fi
  if [ $POUT == 0 -a $NOUT == 4 ];then MT+=(37); fi
  if [ $POUT == 1 -a $NOUT == 2 ];then MT+=(41); fi
  if [ $POUT == 1 -a $NOUT == 3 ];then MT+=(42); fi
  if [ $POUT == 2 -a $NOUT == 1 ];then MT+=(44); fi
  if [ $POUT == 3 -a $NOUT == 3 ];then MT+=(45); fi
  if [ $POUT == 0 -a $NOUT == 0 ];then MT+=(103); fi
  if [ $POUT == 1 -a $NOUT == 0 ];then MT+=(103); fi
  if [ $POUT == 1 -a $NOUT == 1 ];then MT+=(104); fi
  if [ $POUT == 1 -a $NOUT == 2 ];then MT+=(105); fi
  if [ $POUT == 2 -a $NOUT == 1 ];then MT+=(106); fi
  if [ $POUT == 2 -a $NOUT == 2 ];then MT+=(107); fi
  if [ $POUT == 4 -a $NOUT == 4 ];then MT+=(108); fi
  if [ $POUT == 6 -a $NOUT == 6 ];then MT+=(109); fi
  if [ $POUT == 2 -a $NOUT == 0 ];then MT+=(111); fi
  if [ $POUT == 3 -a $NOUT == 2 ];then MT+=(112); fi
  if [ $POUT == 5 -a $NOUT == 6 ];then MT+=(113); fi
  if [ $POUT == 5 -a $NOUT == 5 ];then MT+=(114); fi
  if [ $POUT == 2 -a $NOUT == 1 ];then MT+=(115); fi
```

```
  if [ $POUT == 2 -a $NOUT == 2 ];then MT+=(116); fi
  if [ $POUT == 3 -a $NOUT == 3 ];then MT+=(117); fi
  if [[ -z ${MT} ]] && [[ $POUT -ge 0 ]] && [[ $NOUT -ge 0 ]];then MT+=(5); fi
  MTnum=(${MT[@]}); }


sort_function_underscore () {
  if [ -z $1 ]; then echo -e "ERROR: missing input in 'sort_function_underscore'"; else
  for each in $(echo $1 | sed 's/_/ /g' | tr ' ' '\n' | sort -u); do echo $each; done; fi ; }

sort_function_bar () {
  if [ -z $1 ]; then echo -e "ERROR: missing input in 'sort_function_bar'"; else
  for each in $(echo $1 | sed 's/-/ /g' | tr ' ' '\n' | sort -u); do echo $each; done; fi ; }




############################## DEFINITION OF INPUT ##############################

# CONTROLLING AND RESOLVING PARAMETERS
# Setting output and checking input for product/emission

data_plot=($(sort_function_underscore $1))
input_data_bases=($(sort_function_underscore $2))
input_reaction=($(sort_function_underscore $3))
input_search=($(sort_function_underscore $4))


# Setting output and checking input for plot axis
for each in ${data_plot[@]}; do
  if [[ ${input_reaction} == *x\)* ]]; then
  case $each in
    man ) manual; ;;
    xs | direct | preequilibrium | compound | all-xs ) echo -e "Plotting product cross section $each\n"; ;;
    activity | \#isotopes | yield | production) echo -e "Plotting medical nuclide production $each\n"; ;;
    * ) echo -e "Input cross section '$each' incorrect for product plot\n"; exit 2; ;;
  esac; fi
  if [[ ${input_reaction} != *x\)* ]]; then
  case $each in
    man ) manual; ;;
    xs | yield | all-xs ) echo -e "Plotting emission cross section $each\n"; ;;
    * ) echo -e "Input cross section '$each' incorrect for emission plot\n"; exit 2; ;;
  esac; fi
done


# Setting output and checking input for systems
for each_system in ${input_data_bases[@]}; do
  case $each_system in
  empire | exfor | exper | talys | tendl | gnuplot ) data_bases+=(${each_system}); echo -e "Plotting $
  {each_system^^} data"; ;;
  * ) echo -e "Input '${each_system^^}' for data system to plot incorrect\n"; exit 2; ;;
  esac
done


case $5 in
  edit | no-edit | eps | epslatex | svg | table ) gnuplot_output=$5; ;;
  man ) manual; ;;
  * ) echo -e "Input '$5' incorrect. Choose edit/no-edit/eps/epslatex/svg or 'man' for manual.\n"; exit 2; ;;
esac


case $6 in
  *-* ) gnuplot_x_range=$6; echo; ;;
  *) echo -e "Input '$6' for x-axis range missing. Setting auto scale\n"; ;;
esac




######################### RESOLVING REACTION PARAMETERS #########################

for each_input_reaction in ${input_reaction[@]}; do

  # Setting 'reactions' array for input target and beam
  input_target=($(echo $each_input_reaction | awk -F"(" '{print $1}'))
  input_beam=($(echo $each_input_reaction | awk -F"(" '{print $2}' | awk -F"," '{print $1}'))

  # Setting natural target isotopes
  if [[ ${input_target} == *nat* ]]; then
    natural_conversion ${input_target}
    for (( N = 0; N < ${#PP[*]}; N++ )); do
      temp_targets_PP+=($(echo ${PP[N]}))
      temp_targets_MM+=($(echo nat))
      temp_targets+=($(echo ${AAA[N]}${NZZ}))
    done

  else
  # Non natural target isotopes and
    if [[ ${input_target} == *[0-9][mg][A-Z][a-z] ]]; then
      if [[ ${input_target} == *[0-9][g][A-Z]* ]]; then temp_targets_MM+=($(echo g)); fi
      if [[ ${input_target} == *[0-9][m][A-Z]* ]]; then temp_targets_MM+=($(echo m)); fi
      input_target=$(echo ${input_target} | sed 's/^\(.\{2\}\)\(.\{1\}\)\(.*\)/\1\3/')
    else
      temp_targets_MM+=($(echo n))
    fi
    temp_targets+=(${input_target})
```

```
    temp_targets_PP+=($(echo 1.000000))

  fi

  for (( N = 0; N < ${#temp_targets[*]}; N++ )); do
    temp_beams+=(${input_beam})
  done


  # Setting 'reactions' array for input emission and products
  input_emissions=($(sort_function_bar $(echo $each_input_reaction | awk -F"," '{print $2}' | awk -F")" '{print $1}')))
  input_products=($(sort_function_bar $(echo $each_input_reaction | awk -F"," '{print $2}' | awk -F")" '{print $2}')))

  # Product dependent sequence
  if [[ ${input_emissions[@]} == *x* ]] && [[ ${input_products[@]} != x ]]; then

    if [[ ${input_products[@]} == *all* ]]; then
      # For all products input
      for (( N = 0; N < ${#temp_targets[*]}; N++ )); do
      for (( n = 0; n < 10; n++ )) ; do
      for (( p = 0; p < 5; p++ )) ; do
        nuclide_conversion ${temp_beams[N]}
        BAA=${AAA}; BZZ=${ZZZ}
        nuclide_conversion ${temp_targets[N]}
        TAA=${AAA}; TZZ=${ZZZ}

        AAA=$(($TAA + $BAA - $p - $n)); if [ $AAA -le 1 ]; then AAA=1; fi
        ZZZ=$(($TZZ + $BZZ - $p)); if [ $ZZZ -le 1 ]; then ZZZ=1; fi
        if [ $ZZZ -lt 100 ]; then zZZ="0$ZZZ"; else zZZ="$ZZZ"; fi
        if [ $ZZZ -lt 10 ]; then zZZ="00$ZZZ"; fi
        NZZ=$(awk -v src="$zZZ" '$1 ~ src { print $2 }' $reference)

        temp_products+=(${AAA}${NZZ})
        temp_products_MM+=($(echo NaN))

      done
      done
      done

    else
      # For specific products input
      for each_product in ${input_products[@]}; do
        if [[ ${each_product} == *[0-9][mg][A-Z][a-z] ]]; then
          if [[ ${each_product} == *[0-9][g][A-Z]* ]]; then temp_products_MM+=($(echo g)); fi
          if [[ ${each_product} == *[0-9][m][A-Z]* ]]; then temp_products_MM+=($(echo m)); fi
          each_product=$(echo ${each_product} | sed 's/^\(.\{2\}\)\(.\{1\}\)\(.*\)/\1\3/')
        else
          temp_products_MM+=($(echo NaN))
        fi
        temp_products+=(${each_product})
      done

    fi

    # Resolving reaction arrays
    for (( N = 0; N < ${#temp_targets[*]}; N++ )); do
    for (( M = 0; M < ${#temp_products[*]}; M++ )); do

      # Resolving targets
      nuclide_conversion ${temp_targets[N]}
      targetsAA+=(${AAA}); targetsZZ+=(${ZZZ})
      targetsNN+=(${NNN}); targetsXX+=(${NZZ})
      targetsPP+=(${temp_targets_PP[N]})
      targetsMM+=(${temp_targets_MM[N]})
      targets+=(${temp_targets[N]})

      # Resolving beams
      nuclide_conversion ${temp_beams[N]}
      beamsAA+=(${AAA}); beamsZZ+=(${ZZZ})
      beamsNN+=(${NNN})
      beams+=(${temp_beams[N]})

      # Resolving products
      nuclide_conversion ${temp_products[M]}
      productsAA+=(${AAA}); productsZZ+=(${ZZZ})
      productsNN+=(${NNN}); productsXX+=(${NZZ})
      productsMM+=(${temp_products_MM[M]})
      products+=(${temp_products[M]})

      # Resolving reactions
      reactions+=($(echo ${temp_targets[N]},${temp_beams[N]}))

    done
    done

  else
    # Emission dependent sequence
    # For all emission input
    if [[ $input_emissions != *all* ]] && [[ ${input_products} == x ]]; then
      echo -e "This option is not yet implemented \n"; exit 2

    # For specific emission input
    else
      echo -e "This option is not yet implemented \n"; exit 2
    fi

  fi

done
```

```
493
494
495 echo -e "Reactions to be plotted"
496 for (( N = 0; N < ${#reactions[*]}; N++ )); do
497 printf " ${targets[N]}(${beams[N]},x)${products[N]} \t\t  target ocurrence: %10s \t\t target state: $
    {targetsMM[N]} \t\t product state: ${productsMM[N]}\n" ${targetsPP[N]}
498 done
499
500
501
502
503
504
505 ##################### EMPIRE DATA AND PLOT SEQUENCE ##########################
506 echo -e "\nSorting EMPIRE data"
507
508 # Excluding irrelevant EMPIRE top directories based in reaction product and target
509 for (( N = 0; N < ${#reactions[*]}; N++ )); do
510 for (( M = 0; M < ${#dir_empire[*]}; M++ )); do
511   if [[ ${dir_empire[M]} == *${productsXX[N]}* ]] || [[ ${dir_empire[M]} == *${targetsXX[N]}* ]]; then
512   delete_array+=(${dir_empire[M]}); fi
513 done
514 done
515
516 dir_empire=($(for each in ${delete_array[@]}; do echo $each; done | sort -u)); unset delete_array
517 #for (( N = 0; N < ${#dir_empire[*]}; N++ )); do echo ${dir_empire[N]}; done
518
519
520 # Excluding file duplicities of EMPIRE
521 if [[ ${input_emissions[@]} == *x* ]] && [[ ${data_bases[@]} == *empire* ]]; then
522 temp_reactions=($(for each in ${reactions[@]}; do echo $each; done | sort -u))
523
524
525 # Finding files EMPIRE sequence
526 for each_dir in ${dir_empire[@]}; do
527 for search in ${input_search[@]}; do
528 for (( N = 0; N < ${#temp_reactions[*]}; N++ )); do
529   # Finding files for EMPIRE (product specification inside EMPIRE files)
530   if [[ $data_plot == *xs* ]]; then
531   file_paths_empire+=($(find ${each_dir} -type f -iwholename ${each_dir}\*${temp_reactions[N]}\*${search}\*.xsc
    -not -iwholename \*outputdir\*))
532   fi
533
534   # Finding data files for medical isotope production
535   if [[ $data_plot == production ]] || [[ $data_plot == activity ]] || [[ $data_plot == yield ]] || [[
    $data_plot == \#isotopes ]]; then
536     echo -e "EMPIRE: no support for '$data_plot' data"
537   fi
538
539   if [[ ${#file_paths_empire[*]} == $no_files ]] || [[ ${#file_paths_empire[*]} == 0 ]]; then
540   echo -e "EMPIRE: invalid reaction '${temp_reactions[N]}' and '${search}' in '$(basename ${each_dir})'"
541   else no_files=${#file_paths_empire[*]}; fi
542
543 done
544 done
545 done
546 fi
547
548 file_paths_empire=($(for file_path in ${file_paths_empire[@]}; do echo $file_path; done | sort -u))
549 #for file_path in ${file_paths_empire[@]}; do echo "'$file_path'"; done
550
551
552
553 # Checking if need to run sequense and correct cross section
554 if [ ${#file_paths_empire[*]} != 0 ]; then
555 if [ $data_plot == xs -o $data_plot == all-xs ]; then
556 for (( N = 0; N < ${#file_paths_empire[*]}; N++ )) ; do
557
558   # For product specific reactions. All reactions; resolved in reaction sequence
559   # Sequence based on extraction of all reaction (emission) channels from empire data file
560   if [[ ${input_emissions[@]} == *x* ]] && [[ ${input_products[@]} != *all* ]]; then
561
562   column_no=`cat ${file_paths_empire[$N]} | head -2 | tail -1 | wc -w`
563   for (( C = 8; C < $column_no; C++ )) ; do
564   for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
565     dir_name=$(basename $(dirname ${file_paths_empire[$N]}))
566
567     # Finding the correct products to plot
568     reaction=$(echo ${dir_name} | awk -F"-" '{print $1}')
569     if [[ ${reactions[M]} == ${reaction} ]]; then
570       particle_emission=$(cat ${file_paths_empire[$N]} | head -2 | tail -1 | tr -s ' ' | cut -d ' ' -f$((C+1)) |
    sed 's/)//g; s/(z,//g')
571
572       p-n-emission $particle_emission
573       BAA=${beamsAA[M]}; BZZ=${beamsZZ[M]}
574       TAA=${targetsAA[M]}; TZZ=${targetsZZ[M]}
575
576       AAA=$(($TAA + $BAA - $POUT - $NOUT))
577       ZZZ=$(($TZZ + $BZZ - $POUT))
578       if [ $ZZZ -le 100 ]; then zZZ="0$ZZZ"; else zZZ="$ZZZ"; fi
579       if [ $ZZZ -lt 10 ]; then zZZ="00$ZZZ"; fi
580       NZZ=$(awk -v src="$zZZ" '$1 ~ src { print $2 }' $reference)
581
582       # Excluding irrelevant products
583       if [[ ${products[M]} == ${AAA}${NZZ} ]]; then
584
585         # Setting target with correct reaction abundance
586         if [[ ${input_target[@]} == *nat* ]]; then
587           columns="(\$1):(\$${C}*${targetsPP[M]})"
588         else
589           columns="(\$1):(\$${C})"
```

```
        fi

      [ -n "$cmd_emp" ] && cmd_emp=${cmd_emp}, || cmd_emp=''
      cmd_emp="${cmd_emp} '${file_paths_empire[$N]}' using ${columns} w lp pi 2 lc palette frac 0.$((3 * $N / 5 + 2 * $C)) title 'EMPIRE-$dir_name-${AAA}${NZZ}'"

      fi

    fi

  done
  done
  fi

  if [[ ${input_emissions[@]} != *x* ]]; then
  for (( N = 2; N <= 5; N++ )) ; do
    [ -n "$cmd_emp" ] && cmd_emp=${cmd_emp}, || cmd_emp=''
    dir_name=$(basename $(dirname $(dirname ${file_paths_empire[$N]})))
    xs_name=$(cat ${file_paths_empire[$N]} | head -2 | tail -1 | tr -s ' ' | cut -d ' ' -f$((N+1)))
    cmd_emp="${cmd_emp} '${file_paths_empire[$N]}' using 1:$N w lp pi 2 lc palette frac 0.$((3 * $N / 5)) title 'EMPIRE-${xs_name,,}-$dir_name'"

  done
  fi


done
fi
fi

echo EMPIRE: sorting done




###################### TALYS DATA AND PLOT SEQUENCE ############################
echo -e "\nSorting TALYS data"

# Excluding irrelevant TALYS top directories based in reaction product and target
for (( N = 0; N < ${#reactions[*]}; N++ )); do
for (( M = 0; M < ${#dir_talys[*]}; M++ )); do
  if [[ ${dir_talys[M]} == *${productsXX[N]}* ]] || [[ ${dir_talys[M]} == *${targetsXX[N]}* ]]; then
  delete_array+=(${dir_talys[M]}); fi
done
done

dir_talys=($(for each in ${delete_array[@]}; do echo $each; done | sort -u)); unset delete_array
#for (( N = 0; N < ${#dir_talys[*]}; N++ )); do echo ${dir_talys[N]}; done


# Finding files TALYS sequence
if [[ ${input_emissions[@]} == *x* ]] && [[ ${data_bases[@]} == *talys* ]]; then
for each_dir in ${dir_talys[@]}; do
for search in ${input_search[@]}; do
for (( N = 0; N < ${#reactions[*]}; N++ )); do

  # Finding product specific, natural target, files
  if [[ ${input_target[@]} == *nat* ]]; then
    search_target=$(echo nat${targetsXX[N]}); search_beam=$(echo ${beams[N]})
  else
    search_target=$(echo ${targets[N]}); search_beam=$(echo ${beams[N]})
  fi

  # Finding data files for medical isotope production
  if [[ $data_plot == production ]] || [[ $data_plot == activity ]] || [[ $data_plot == yield ]] || [[ $data_plot == \#isotopes ]]; then
    file_paths_talys+=($(find ${each_dir} -type f -iwholename \*${search_target},${search_beam}\*${search}\*/Y\*${productsZZ[N]}\*${productsAA[N]}\*.\* -not -name \*prev\*))
  fi

  # Finding product specific files
  if [[ $data_plot == xs ]]; then
  file_paths_talys+=($(find ${each_dir} -type f -iwholename \*${search_target},${search_beam}\*${search}\*/rp\*${productsZZ[N]}\*${productsAA[N]}.\* -not -name \*prev\*))
  fi

  if [[ ${#file_paths_talys[*]} == $no_files ]] || [[ ${#file_paths_talys[*]} == 0 ]]; then
  echo -e "TALYS: invalid $data_plot product '${products[N]}' for reaction '${reactions[N]}' and '${search}' in '$(basename ${each_dir})'"
  else no_files=${#file_paths_talys[*]}; fi

done
done
done
fi

file_paths_talys=($(for file_path in ${file_paths_talys[@]}; do echo $file_path; done | sort -u))
#for file_path in ${file_paths_talys[@]}; do echo "'$file_path'"; done


# Checking if need to run sequense
if [ ${#file_paths_talys[*]} != 0 ]; then

# Defining plotting function for TALYS
cmd_talys () {
  # Resolving reaction parameters from file to get correct plot titles
  dir_name=$(basename $(dirname $(dirname ${file_paths_talys[$N]})))
  base_name=$(basename ${file_paths_talys[$N]} | sed 's/.tot//g')
  product=$(echo $base_name | sed 's/^[A-Z]*//g;s/^[a-z]*//g' | awk -F"." '{print $1}')
  nuclide_conversion ${product}
```

```
685
686   # Setting target with correct title/plot for natural targets
687   target=$(echo $dir_name | awk -F"-" '{print $1}' | awk -F"," '{print $1}')
688   base_target=$(echo $base_name | sed 's/.L[0-9][0-9]//' | awk -F"." '{print $2}' | sed 's/^0*//')
689   if [[ ! -z $base_target ]]; then
690     target="${target/nat/nat(${base_target})}"      # Replaces nat with $base_target
691     dir_name="${dir_name/nat/nat(${base_target})}"  # Replaces nat with $base_target
692   fi
693
694   # Setting product with correct title/plot for ground/meta state
695   if [[ ${base_name} == *.L[0-9][0-9]* ]] && [[ ${productsMM[@]} != *[g-m]* ]]; then
696     continue
697   else
698     if [[ $base_name == *[0-9].L00* ]]; then state="g"; fi
699     if [[ $base_name == *[0-9].L[0-9][1-9] ]]; then state="m"; fi
700   fi
701
702   # Setting target with correct reaction abundance
703   for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
704     if [[ "nat(${targetsAA[$M]})${targetsXX[$M]}" == ${target} ]]; then targetPP=${targetsPP[M]}; fi
705     if [[ "nat${targetsXX[$M]}" == ${target} ]]; then targetPP=($(echo 1.000000)); fi
706     if [[ ${targets[$M]} == ${target} ]]; then targetPP=($(echo 1.000000)); fi
707   done
708
709   if [ $data_plot != all-xs -a $data_plot != production ]; then
710     if [ $data_plot != xs ]; then data_plot=${data_plot^}; fi
711     column_no=$(awk -v src=${data_plot} '{ for(i=1;i<=NF;i++){if ($i ~ src) {print i } } }' $
    {file_paths_talys[$N]})
712     if [[ ${file_paths_talys[$N]} == *Y*.tot ]]; then column_no=$(( column_no / 2)); else column_no=$((
    column_no - 1)); fi
713   fi
714
715   # Setting correct column to plot in files
716   if [[ ${file_paths_talys[$N]} == *Y*.tot ]]; then
717     column_name=$(cat ${file_paths_talys[$N]} | head -6 | tail -1 | tr -s ' ' | cut -d ' ' -f$(( column_no * 2
    )))
718     column_names=$(cat ${file_paths_talys[$N]} | head -6 | tail -1 | tr -s ' ' | cut -d ' ' -f$(( column_no * 2
    + 1)))
719     ylabel="$column_name $column_names"
720   else
721     column_name=$(cat ${file_paths_talys[$N]} | head -5 | tail -1 | tr -s ' ' | cut -d ' ' -f$(( column_no +
    1)))
722   fi
723
724   # Setting final variables
725   columns="(\$1):(\$${column_no}*$targetPP)"
726   title=$(echo TALYS-${column_name,,}-$dir_name-${AAA}${state}${NZZ})
727   [ -n "$cmd_tal" ] && cmd_tal=${cmd_tal}, || cmd_tal=''
728   cmd_tal="${cmd_tal} '${file_paths_talys[$N]}' using ${columns} w lp pi 2 lc palette frac 0.$((3 + 3 * $N / 5))
    title '${title}'"; }
729
730
731 for (( N = 0; N < ${#file_paths_talys[*]}; N++ )) ; do
732
733   # Plotting all cross sections/medical isotope production choises
734   if [ $data_plot == all-xs -o $data_plot == production ]; then
735     if [ $data_plot == production ]; then MAXC=5; fi
736     if [ $data_plot == all-xs ]; then MAXC=6; fi
737     for (( column_no = 2; column_no < $MAXC ; column_no++ )) ; do cmd_talys; done
738
739   # Plotting specific cross section/medical isotope production choise
740   else
741     cmd_talys
742   fi
743
744 done
745 fi
746
747 echo TALYS: sorting done
748
749
750
751
752
753 ##################### TENDL DATA AND PLOT SEQUENCE ############################
754 echo -e "\nSorting TENDL data"
755
756 # Excluding irrelevant TALYS top directories based in reaction product and target
757 for (( N = 0; N < ${#reactions[*]}; N++ )); do
758 for (( M = 0; M < ${#dir_tendl[*]}; M++ )); do
759   if [[ ${dir_tendl[M]} == *${productsXX[N]}* ]] || [[ ${dir_tendl[M]} == *${targetsXX[N]}* ]]; then
760   delete_array+=(${dir_tendl[M]}); break; fi
761 done
762 done
763
764 dir_tendl=($(for each in ${delete_array[@]}; do echo $each; done | sort -u)); unset delete_array
765 #for (( N = 0; N < ${#dir_tendl[*]}; N++ )); do echo ${dir_tendl[N]}; done
766
767
768 # Finding files TENDL sequence
769 if [[ ${input_emissions[@]} == *x* ]] && [[ ${data_bases[@]} == *tendl* ]]; then
770 for each_dir in ${dir_tendl[@]}; do
771 for (( N = 0; N < ${#reactions[*]}; N++ )); do
772
773   # Finding product specific files
774   if [[ $data_plot == xs ]]; then
775     file_paths_tendl+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*${reactions[N]}/\*${products[N]}
    \*.tot))
776   fi
777
778   # Finding data files for medical isotope production
```

```
  if [[ $data_plot == production ]] || [[ $data_plot == activity ]] || [[ $data_plot == nuclides ]]; then
    file_paths_tendl+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*${reactions[N]}/Y\*${products[N]}\*.tot))
  fi

  if [[ ${#file_paths_tendl[*]} == $no_files ]] || [[ ${#file_paths_tendl[*]} == 0 ]]; then
  echo -e "TENDL: invalid $data_plot product '${products[N]}' for reaction '${reactions[N]}'"
  else no_files=${#file_paths_tendl[*]}; fi

done
done
fi

file_paths_tendl=($(for file_path in ${file_paths_tendl[@]}; do echo $file_path; done | sort -u))
#for file_path in ${file_paths_tendl[@]}; do echo "'$file_path'"; done



# Checking if need to run sequense and correct cross section xs or all-xs
if [[ $data_plot == *xs* ]] && [[ ${data_bases[@]} == *tendl* ]]; then
if [ ${#file_paths_tendl[*]} != 0 ]; then
for (( N = 0; N < ${#file_paths_tendl[*]}; N++ )) ; do

  # Resolving reaction parameters from file to get correct plot titles
  dir_name=$(basename $(dirname $(dirname ${file_paths_tendl[$N]})))
  base_name=$(basename ${file_paths_tendl[$N]} | sed 's/.tot//g')

  # Setting target with correct reaction abundance
  target=$(echo $base_name | awk -F"," '{print $2}' | awk -F"(" '{print $1}' | sed 's/ //g')
  for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
    if [[ ${targets[$M]} == ${target} ]]; then
      targetPP=${targetsPP[M]}
      break
      fi
  done

  # Setting product with correct title/plot for ground/meta state
  if [[ ${base_name} == *[0-9[0-9][g-m][A-Z][a-z]* ]] && [[ ${productsMM[@]} != *[g-m]* ]]; then
    continue
  fi

  [ -n "$cmd_ten" ] && cmd_ten=${cmd_ten}, || cmd_ten=''
  columns="(\$1):(\$2*$targetPP) w lp pi 1"
  cmd_ten="${cmd_ten} '${file_paths_tendl[$N]}' using ${columns} lc palette frac 0.$((6 + 3 * $N / 5)) title '$base_name'"

done
fi
fi

echo TENDL: sorting done




##################### EXFOR DATA AND PLOT SEQUENCE ############################
echo -e "\nSorting EXFOR data"

# Finding files EXFOR sequence
if [[ ${input_emissions[@]} == *x* ]] && [[ $data_plot == xs ]] && [[ ${data_bases[@]} == *exfor* ]]; then
for each_dir in ${dir_exfor}; do
for (( N = 0; N < ${#reactions[*]}; N++ )); do

  # Finding product specific files
  if [[ ${beams[N]} != *[0-9]* ]]; then
    file_paths_exfor+=($(find $dir_exfor -type f -iwholename $dir_exfor/${beams[N]}\*/\*${targetsZZ[N]}\*${targetsXX[N]}\*${targetsAA[N]}.c4))
  else
    file_paths_exfor+=($(find $dir_exfor -type f -iwholename $dir_exfor/other/\*${targetsZZ[N]}\*${targetsXX[N]}\*${targetsAA[N]}.c4))
  fi

  if [[ ${#file_paths_exfor[*]} == $no_files ]] || [[ ${#file_paths_exfor[*]} == 0 ]]; then
    echo -e "EXFOR: invalid $data_plot product '${products[N]}' for reaction '${reactions[N]}'"
  else no_files=${#file_paths_exfor[*]}; fi

done
done
fi

file_paths_exfor=($(for file_path in ${file_paths_exfor[@]}; do echo $file_path; done | sort -u))
#for file_path in ${file_paths_exfor[@]}; do echo "'$file_path'"; done



# EXFOR SEQUENCE
# Checking if need to run sequense and correct cross section
if [[ $data_plot == *xs* ]] && [[ ${data_bases[@]} == *exfor* ]]; then
if [[ ${#file_paths_exfor[*]} != 0 ]]; then
for (( N = 0; N < ${#file_paths_exfor[*]}; N++ )) ; do

# Copying file so that the official EXFOR file keeps intact
cp ${file_paths_exfor[$N]} $gnuplot_temp
file_paths_exfor[$N]=$gnuplot_temp/$(basename ${file_paths_exfor[$N]})

# Producing plots for all products
for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
  NOUT=$((${targetsNN[M]} + ${beamsNN[M]} - ${productsNN[M]}))
  POUT=$((${targetsZZ[M]} + ${beamsZZ[M]} - ${productsZZ[M]}))

```

```
  # Number of individual data sets in EXFOR file
  # each_MT_sets is the first line for each data set
  MT_set=(); MT_number
  for each_MT in ${MTnum[@]}; do
    MT_set+=($(grep -n " "$each_MT"[[:alpha:]]*[[:blank:]]*]" ${file_paths_exfor[$N]} | grep "C4" | awk -F":" '{print $1}'))
  done

  for each_MT_set in ${MT_set[@]};do
    each_MT_linei=$(($each_MT_set + 4)) # First line of data to plot in each data set
    # Last line of data to plot in each data set. Taken from listen number of data sets in EXFOR file
    each_MT_linef=$(($(sed -n $(($each_MT_linei - 3))p ${file_paths_exfor[$N]}| awk '{print $2}') + $each_MT_linei))
    # Lines omitted by gnuplot and corrected counting from zero
    unused_lines=$(($(head -n ${each_MT_linei} ${file_paths_exfor[$N]} | grep "\#" | wc -l) + 1))

    # Omitting irrelevant data sets
    not_xs_data_set=$(sed -n "$each_MT_linei"p ${file_paths_exfor[$N]} | sed 's/^\(.\{59\}\)\(.\{25\}\)\(.*\)/\2/')

    if [[ -z "${not_xs_data_set// }" ]]; then
    for (( P = $each_MT_linei; P < ${each_MT_linef}; P++ )); do
      dataline=$(sed -n ${P}p ${file_paths_exfor[$N]})

      # Setting correct scientific format for data column 8
      column08pm=$(echo ${dataline} | sed 's/^\(.\{56\}\)\(.\{1\}\)\(.*\)/\2/')
      if [[ ${column08pm// } == + ]] || [[ ${column08pm// } == - ]]; then
        sed "${P} s/^\(.\{56\}\)\(.\{1\}\)\(.*\)/\1E\2\3/" -i ${file_paths_exfor[$N]}
      else
        sed "${P} s/^\(.\{58\}\)\(.\{1\}\)\(.*\)/\1  \3/" -i ${file_paths_exfor[$N]}
      fi

      # Setting empty value in column 7 to zero
      column07=$(echo ${dataline} | sed 's/^\(.\{50\}\)\(.\{8\}\)\(.*\)/\2/')
      if [[ -z "${column07// }" ]]; then
      sed "${P} s/^\(.\{50\}\)\(.\{1\}\)\(.*\)/\10\3/" -i ${file_paths_exfor[$N]}; fi

      # setting correct scientific format for data column 7
      column07pm=$(echo ${dataline} | sed 's/^\(.\{47\}\)\(.\{1\}\)\(.*\)/\2/')
      if [[ ${column07pm// } == + ]] || [[ ${column07pm// } == - ]]; then
        sed "${P} s/^\(.\{47\}\)\(.\{1\}\)\(.*\)/\1E\2\3/" -i ${file_paths_exfor[$N]}
      else
        sed "${P} s/^\(.\{49\}\)\(.\{1\}\)\(.*\)/\1\2\2\3/" -i ${file_paths_exfor[$N]}
      fi

      # Setting correct scientific format for data column 6
      column06pm=$(echo ${dataline} | sed 's/^\(.\{38\}\)\(.\{1\}\)\(.*\)/\2/')
      if [[ ${column06pm// } == + ]] || [[ ${column06pm// } == - ]]; then
        sed "${P} s/^\(.\{38\}\)\(.\{1\}\)\(.*\)/\1E\2\3/" -i ${file_paths_exfor[$N]}
      else
        sed "${P} s/^\(.\{40\}\)\(.\{1\}\)\(.*\)/\1\2\2\3/" -i ${file_paths_exfor[$N]}
      fi

      # Setting empty value in column 5 to zero
      column05=$(echo ${dataline} | sed 's/^\(.\{32\}\)\(.\{8\}\)\(.*\)/\2/')
      if [[ -z "${column05// }" ]]; then
        sed "${P} s/^\(.\{32\}\)\(.\{1\}\)\(.*\)/\10\3/" -i ${file_paths_exfor[$N]}; fi

      # setting correct scientific format for data column 5
      column05pm=$(echo ${dataline} | sed 's/^\(.\{29\}\)\(.\{1\}\)\(.*\)/\2/')
      if [[ ${column05pm// } == + ]] || [[ ${column05pm// } == - ]]; then
        sed "${P} s/^\(.\{29\}\)\(.\{1\}\)\(.*\)/\1E\2\3/" -i ${file_paths_exfor[$N]}
      else
        sed "${P} s/^\(.\{31\}\)\(.\{1\}\)\(.*\)/\1\2\2\3/" -i ${file_paths_exfor[$N]}
      fi

    done

    # Energy plotted to Cross section; with energy error lower and upper limit ; with cross section error lower and upper limit
    # Setting target with correct reaction abundance
    columns="(\$5*1E-6):(\$7*1E+3*${targetsPP[M]}):(\$6*1E-6):(\$8*1E+3*${targetsPP[M]}) w xyerrorbars"
    every="every ::$(($each_MT_linei - $unused_lines))::$(($each_MT_linef - $unused_lines))"

    AUTHOR=$(sed -n "$each_MT_linei"p ${file_paths_exfor[$N]} | sed 's/^\(.\{98\}\)\(.*\)/\2/' | awk  -F"(" '{print $1}')
    AUTHOR=$(echo ${AUTHOR[@],,} | sed -r 's/\<./\U&/g' | sed 's/([0-9]*)//g;s/[[:space:]]//g;s/\./\. /g;s/,Et. Al./ et al./g' )
    YEAR=$(sed -n "$each_MT_linei"p ${file_paths_exfor[$N]} | awk  -F")" '{print $1}' | awk  -F"(" '{print $2}')
    if [[ $YEAR > 20 ]]; then YEAR=$(echo 19${YEAR}); else YEAR=$(echo 20${YEAR}); fi
    TITLE=$(echo ${AUTHOR} \(${YEAR}\), ${targets[M]}\(${beams[M]},${NOUT}n${POUT}p\)${products[M]})
    TITLE=$(echo $TITLE | sed 's/  / /g;s/0p//g;s/1p/p/g;s/0n//g;s/1n/n/g;s/protons/p/g;s/alphas/a/g;s/deutrons/d/g;s/helions/h/g;')

    # Plotting
    if [ -n "${columns}" ]; then
      [ -n "$cmd_exf" ] && cmd_exf=${cmd_exf}, || cmd_exf=''
      cmd_exf="${cmd_exf} '${file_paths_exfor[$N]}' ${every} using ${columns} lc palette frac 0.$((0 + 4 * $P / 2)) title '$TITLE'"
    fi

    fi
  done

  done
  done

fi
fi

echo EXFOR: sorting done
```

```
969
970
971
972
973
974 ######################## EXPER DATA AND PLOT SEQUENCE ########################
975 echo -e "\nSorting EXPER data"
976
977 # Finding files EXPER/EXFOR sequence
978 if [[ ${data_bases[@]} == *exper* ]]; then
979 for each_dir in ${dir_exper}; do
980 for (( N = 0; N < ${#reactions[*]}; N++ )); do
981
982   # Finding product specific files
983   if [[ $data_plot == xs ]]; then
984     if [[ ${targetsMM[N]} == nat ]]; then
985       file_paths_exper+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*nat${targetsXX[N]}\*${beams[N]}/\*
      ${products[N]}\*))
986     else
987       file_paths_exper+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*${reactions[N]}/\*${products[N]}
      \*))
988     fi
989   fi
990
991   # Finding data files for medical isotope production
992   if [[ $data_plot == production ]] || [[ $data_plot == activity ]] || [[ $data_plot == nuclides ]]; then
993     file_paths_exper+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*${reactions[N]}/Y\*${products[N]}
    \*))
994   fi
995
996   if [[ ${#file_paths_exper[*]} == $no_files ]] || [[ ${#file_paths_exper[*]} == 0 ]]; then
997   echo -e "EXPER: invalid $data_plot product '${products[N]}' for reaction '${reactions[N]}'"
998   else no_files=${#file_paths_exper[*]}; fi
999
1000 done
1001 done
1002 fi
1003
1004 file_paths_exper=($(for file_path in ${file_paths_exper[@]}; do echo $file_path; done | sort -u))
1005 #for file_path in ${file_paths_exper[@]}; do echo "'$file_path'"; done
1006
1007
1008
1009 # EXPER/EXFOR SEQUENCE
1010 # Checking if need to run sequense and correct cross section xs or all-xs
1011 if [[ $data_plot == *xs* ]] && [[ ${data_bases[@]} == *exper* ]]; then
1012 if [ ${#file_paths_exper[*]} != 0 ]; then
1013 for (( N = 0; N < ${#file_paths_exper[*]}; N++ )) ; do
1014
1015   # Resolving reaction parameters from file to get correct plot titles
1016   dir_name=$(basename $(dirname $(dirname ${file_paths_exper[$N]})))
1017   base_name=$(basename ${file_paths_exper[$N]} | sed 's/.tot//g;s/.x4//g')
1018
1019   # Setting target with correct reaction abundance
1020   target=$(echo $base_name | awk -F"," '{print $2}' | awk -F"(" '{print $1}' | sed 's/ //g')
1021   for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
1022     if [[ ${targets[$M]} == ${target} ]]; then
1023       targetPP=${targetsPP[M]}; break
1024     else
1025       targetPP=$((1))
1026     fi
1027   done
1028
1029   [ -n "$cmd_exp" ] && cmd_exp=${cmd_exp}, || cmd_exp=''
1030   columns="(\$1*1E-6):(\$4*1E+3*$targetPP):((\$1*1E-6)-(\$2*1E-6)):((\$1*1E-6)+(\$3*1E-6)):(((\$4*1E+3)-(\$5*1E
    +3))*$targetPP):(((\$4*1E+3)+(\$6*1E+3))*$targetPP) w xyerrorbars"
1031   cmd_exp="${cmd_exp} '${file_paths_exper[$N]}' using ${columns} lc palette frac 0.$((8 + 3 * $N / 5)) title
    '$base_name'"
1032
1033 done
1034 fi
1035 fi
1036
1037 echo EXPER: sorting done
1038
1039
1040
1041
1042
1043 ######################## GNUPLOT DATA AND PLOT SEQUENCE ########################
1044 echo -e "\nSorting GNUPLOT data"
1045
1046 # Finding files GNUPLOT sequence
1047 if [[ ${data_bases[@]} == *gnuplot* ]]; then
1048 for each_dir in ${dir_gnuplot[@]}; do
1049 for (( N = 0; N < ${#reactions[*]}; N++ )); do
1050
1051   # Finding product specific files
1052   if [[ $data_plot == xs ]] && [[ ${input_emissions[@]} == *x* ]]; then
1053   file_paths_gnuplot+=($(find ${each_dir} -type f -iwholename ${each_dir}/\*${reactions[N]}/\*${products[N]}\*))
1054   fi
1055
1056   if [[ ${#file_paths_gnuplot[*]} == $no_files ]] || [[ ${#file_paths_gnuplot[*]} == 0 ]]; then
1057   echo -e "GNUPLOT: invalid $data_plot product '${products[N]}' for reaction '${reactions[N]}'"
1058   else no_files=${#file_paths_gnuplot[*]}; fi
1059
1060 done
1061 done
1062 fi
1063
1064 file_paths_gnuplot=($(for file_path in ${file_paths_gnuplot[@]}; do echo $file_path; done | sort -u))
```

```
#for file_path in ${file_paths_gnuplot[@]}; do echo "'$file_path'"; done



# Checking if need to run sequense and correct cross section xs or all-xs
if [[ $data_plot == *xs* ]] && [[ ${data_bases[@]} == *gnuplot* ]]; then
if [ ${#file_paths_gnuplot[*]} != 0 ]; then
for (( N = 0; N < ${#file_paths_gnuplot[*]}; N++ )) ; do

  # Resolving reaction parameters from file to get correct plot titles
  dir_name=$(basename $(dirname $(dirname ${file_paths_gnuplot[$N]})))
  base_name=$(basename ${file_paths_gnuplot[$N]} | sed 's/.tot//g')

  # Setting target with correct reaction abundance
  target=$(echo $base_name | awk -F"," '{print $2}' | awk -F"(" '{print $1}' | sed 's/ //g')
  for (( M = 0; M < ${#reactions[*]}; M++ )) ; do
    if [[ ${targets[$M]} == ${target} ]]; then
      targetPP=${targetsPP[M]}; break
    else
      targetPP=$((1))
    fi
  done

  [ -n "$cmd_gnu" ] && cmd_gnu=${cmd_gnu}, || cmd_gnu=''
  columns="(\$1):(\$2*$targetPP) w lp pi 1"
  cmd_gnu="${cmd_gnu} '${file_paths_gnuplot[$N]}' using ${columns} lc palette frac 0.$((9 + 3 * $N / 5)) title
  '$base_name'"

done
fi
fi

echo GNUPLOT: sorting done




######################### GNUPLOT PLOTLINE PARAMETERS #########################

# Checking plottable files, printing statistics
echo
echo -e "Found $(echo $cmd_emp | grep -o "title" | wc -l) EMPIRE data series to plot"
echo -e "Found $(echo $cmd_tal | grep -o "title" | wc -l) TALYS data series to plot"
echo -e "Found $(echo $cmd_ten | grep -o "title" | wc -l) TENDL data series to plot"
echo -e "Found $(echo $cmd_exf | grep -o "title" | wc -l) EXFOR data series to plot"
echo -e "Found $(echo $cmd_exp | grep -o "title" | wc -l) EXPER/EXFOR data series to plot"
echo -e "Found $(echo $cmd_gnu | grep -o "title" | wc -l) GNUPLOT data series to plot\n"


# MANAGING TITLES
# Managing plot titles
TIME=`date '+%Y.%m.%d-%H.%M'`
plot="Cross section data | ${data_plot[@]} | ${input_reaction[@]} | ${input_data_bases[@]^^} | $TIME"
plot_title="Cross section data: ${input_reaction[@]}| ${input_data_bases[@]^^} $data_plot cross section(s)"

# Managing plot labels and titels
if [[ ${input_reaction} == *x* ]]; then
data_plot=${data_plot,}
  if [ $data_plot == production -o $data_plot == activity -o $data_plot == yield -o $data_plot == \#isotopes ];
  then
    xlabel="Time (h)"
    plot=$(echo $plot | sed 's/Cross section data/Nuclide production data/g')
    plot_title=$(echo $plot_title | sed 's/Cross section data/Nuclide production data/g')
    plot_title=$(echo $plot_title | sed 's/cross section(s)//g')

  else
  xlabel="Energy (MeV)"; ylabel="Cross section (mb)"
  fi

else
  xlabel="Emission energy (MeV)"; ylabel="Cross section (mb)"
fi

# Cleaning plot titles
plot=$(echo $plot | sed 's/_/ /g;s/-/,/g;s/|/-/g' | sed -r 's/,+/,/g')
plot_title=$(echo $plot_title | sed 's/_/ /g;s/-/,/g;s/|/,/g' | sed -r 's/,+/,/g')



# MERGING CMD COMMANDS
# Merging commands for theoretical and experimental plots
if [ ! -z $cmd_emp ]; then cmd=($cmd_emp); fi
if [ ! -z $cmd_tal ]; then
  if [ -z $cmd ]; then cmd=($cmd_tal)
  else cmd=$(echo $cmd, $cmd_tal); fi; fi
if [ ! -z $cmd_ten ]; then
  if [ -z $cmd ]; then cmd=($cmd_ten)
  else cmd=$(echo $cmd, $cmd_ten); fi; fi
if [ ! -z $cmd_exf ]; then
  if [ -z $cmd ]; then cmd=($cmd_exf)
  else cmd=$(echo $cmd, $cmd_exf); fi; fi
if [ ! -z $cmd_exp ]; then
  if [ -z $cmd ]; then cmd=($cmd_exp)
  else cmd=$(echo $cmd, $cmd_exp); fi; fi
if [ ! -z $cmd_gnu ]; then
  if [ -z $cmd ]; then cmd=($cmd_gnu)
  else cmd=$(echo $cmd, $cmd_gnu); fi; fi


```

```
1164
1165
1166 # Checking for errors in cmd
1167 if [ -z $cmd ]; then
1168   echo -e "Found no data to plot. Check reaction parameters. \n"; exit 2
1169   else echo -e "Plotting..."
1170 fi
1171
1172
1173
1174
1175 # GENERATING GNUPLOT SCRIPT
1176 # Setting Gnuplot script parameter for printing
1177 if [ $gnuplot_output == eps -o $gnuplot_output == epslatex -o $gnuplot_output == svg -o $gnuplot_output == table
    ]; then
1178   if [ $gnuplot_output == eps ]; then
1179   command_print+=("set terminal postscript enhanced color font 'lmodern,10'
1180   set output '$gnuplot_plot/$plot.eps'
1181   replot
1182   ");fi
1183
1184   if [ $gnuplot_output == epslatex ]; then
1185   command_print+=("set terminal epslatex size 8.0,4.0 color colortext
1186   unset title
1187   set output '$gnuplot_plot/$plot.tex'
1188   replot
1189   ");fi
1190
1191   if [ $gnuplot_output == svg ]; then
1192   command_print+=("set terminal svg size 800,400 fname 'lmodern' fsize 10
1193   unset title
1194   set output '$gnuplot_plot/$plot.svg'
1195   replot
1196   ");fi
1197
1198   if [ $gnuplot_output == table ]; then
1199   command_print+=("set table
1200   set output '$gnuplot_plot/$plot.table'
1201   replot
1202   ");fi
1203
1204 else
1205 command_print="#set terminal svg size 800,400 fname 'lmodern' fsize 10
1206 #set output '$gnuplot_plot/$plot.svg'
1207 #replot"
1208 fi
1209
1210 if [ -z $gnuplot_x_range ]; then
1211   xrange="#set xrange [ 0 : 40 ]"
1212 else
1213   x_min=`echo $gnuplot_x_range | awk -F"-" '{print $1}'`
1214   x_max=`echo $gnuplot_x_range | awk -F"-" '{print $2}'`
1215   xrange="set xrange [ $x_min : $x_max ]"
1216 fi
1217
1218 # Writing Gnuplot script and parameters
1219 cat <<EOF > $gnuplot_temp/plotfile.gnu
1220 set terminal wxt size 800,400 persist
1221 set autoscale                                       # scale axes automatically
1222 $xrange                                             # scale x-axis
1223 unset log                                           # remove any log-scaling
1224 unset label                                         # remove any previous labels
1225 set xtic auto                                       # set xtics automatically
1226 set ytic auto                                       # set ytics automatically
1227
1228 set title "$plot_title"
1229 set xlabel "$xlabel"
1230 set ylabel "$ylabel"
1231 set key outside left bottom horizontal center Left reverse
1232 set key noinvert samplen 0.2 spacing 1 width 0 height 2
1233 unset colorbox
1234 # MATLAB jet color palette
1235 # line styles
1236 set style line 1 lt 1 lc rgb '#000080' #
1237 set style line 2 lt 1 lc rgb '#0000ff' #
1238 set style line 3 lt 1 lc rgb '#0080ff' #
1239 set style line 4 lt 1 lc rgb '#00ffff' #
1240 set style line 5 lt 1 lc rgb '#80ff80' #
1241 set style line 6 lt 1 lc rgb '#ffff00' #
1242 set style line 7 lt 1 lc rgb '#ff8000' #
1243 set style line 8 lt 1 lc rgb '#ff0000' #
1244 set style line 9 lt 1 lc rgb '#800000' #
1245 # palette
1246 set palette defined (0 0.0 0.0 0.5, 1 0.0 0.0 1.0, 2 0.0 0.5 1.0, 3 0.0 1.0 1.0, \
1247 4 0.5 1.0 0.5, 5 1.0 1.0 0.0, 6 1.0 0.5 0.0, 7 1.0 0.0 0.0, 8 0.5 0.0 0.0 )
1248
1249 plot $cmd
1250
1251 set table
1252 set output '$gnuplot_temp/data-table.table'
1253 replot
1254
1255 unset table
1256 replot
1257
1258 #In case for building an output file
1259 ${command_print[@]}
1260 EOF
1261
1262
1263 # Writing Gnuplot plotline
```

```
cat <<EOF > $gnuplot_temp/plotline.gnu
plot $cmd
EOF


# Run plot command
case $gnuplot_output in
  edit ) gnuplot $gnuplot_temp/plotfile.gnu - ; ;;
  no-edit | eps | epslatex | svg | table ) gnuplot $gnuplot_temp/plotfile.gnu ; ;;
esac

echo -e "\nScript finished\n\n"
unset IFS


```