1 Default Algorithms

Algorithm 1 Data conversion from NIfTI to .csv

Input: $Vol_{ijkt} \leftarrow$ 4-D double matrix data for $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$, $t = (1, ..., T)$ from a NIfTI .nii file.
Output: Reduced data in a comma-separated values .csv file.

$file \leftarrow newFile()$
$aux \leftarrow Vol.removeEvenCoordinates()$
for $i = 1$ to $Vol.lengthOfDimension(x)$ **do**
  for $j = 1$ to $Vol.lengthOfDimension(y)$ **do**
    for $k = 1$ to $Vol.lengthOfDimension(z)$ **do**
      for $t = 1$ to T **do**
        if $t = 1$ **then**
          $metabolism \leftarrow aux_{ijkt}$
        else
          $metabolism \leftarrow ',' + aux_{ijkt}$
        end if
      end for
      $line \leftarrow i + ',' + ',' + j + ' k' + ',' + metabolism$
      $file.write(line)$
    end for
  end for
end for

Algorithm 2 Average by section

Input:

$file \leftarrow$ Data in Comma-Separated Values .csv format file with header i,j,k,t, $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$, $t = (1, ..., T)$.

$range_{mn} \leftarrow$ array containing the ranges boundary values for each m section.

Output: Section average value and average volume value given time in a comma-separated values .csv file.

$newFile \leftarrow createFile("filename")$
$aux \leftarrow Vol.removeEvenCoordinates()$
for $i = 1$ to m **do**
    for $j = 1$ to n **do**
        $avg \leftarrow averageBySection(file, ranges_{m0}, ranges_{m1}, ranges_{m2}, ranges_{m3}, ranges_{m4}, ranges_{m5})$
        if $i = 0$ **then**
            $averages \leftarrow avg$
        else
            $averages \leftarrow ',' + avg$
        end if
    end for
end for
$averages \leftarrow ',' + average(file)$
$newfile.write(averages)$

{$averageBySection()$ calculates the average value within the specified range values for each time, meanwhile $average()$ calculates the average metabolism value of the whole volume for each time.}

Algorithm 3 Level asignation given voxel value

Input:

$file \leftarrow$ Path to data file in comma-separated values .csv file with header i,j,k,t, $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$, $t = (1, ..., T)$.

$array3D \leftarrow$ An empty 3-D array.

$t \leftarrow$ The time picked to assign values to the 3-D Array

Output: A 3-D Array with assigned level values given spatial position $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$ in a certain time t.

$file \leftarrow readFile(file)$
for all $line$ in $lines$ **do**
    $x \leftarrow line_0$
    $y \leftarrow line_1$
    $z \leftarrow line_2$
    $rangevalue \leftarrow setRange(line_t)$
    $array3D_{x,y,z} \leftarrow rangevalue$
end for
return $array3D_{x,y,z}$

{$setRange()$ assigns a value given the metabolism value by the table range criteria. $readFile()$ splits the .csv lines.}

Algorithm 4 Connection of high energy level voxels

Input: $array3D_{ijk} \leftarrow$ 3-D Array with assigned level values given spatial position $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$

Output: High level connections in set of pairs of positions by level.

```
candidates ← selectPositionsWhere(array3D == level)
for all voxel in candidates do
    minimum distance ← high initial value
    y ← line_1
    for all other voxel in candidates do
        if other voxel is not voxel then
            distance ← euclideanDistance(voxel, other voxel)
            if distance < minimum distance then
                if pairs.isEmpty() then
                    minimum distance ← distance
                    minimum ← other voxel
                end if
                for all pair in pairs do
                    if not other voxel.equals(pair_0) and other voxel
                    not in selected candidates then
                        minimum distance ← distance
                        minimum ← other voxel
                    end if
                end for
            end if
        end if
    end for
    if minimum is not None then
        new pair ← [voxel, minimum]
        selected candidates.append(voxel)
        pairs.append(new pair)
    end if
end for
return pairs
```

Algorithm 5 Connection between voxels

Input: $array3D_{ijk} \leftarrow$ 3-D Array with assigned level values given spatial position $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$.
$connections \leftarrow$ Empty array for pairs of positions.
Output: Connections in set of pairs of positions.

$file \leftarrow readFile(file)$
for all i in $range(1, x)$ **do**
  for all j in $range(1, y)$ **do**
    for all k in $range(1, z)$ **do**
      $current\ value \leftarrow array3D_{ijk}$
      if $current\ value = 8$ **or** $current\ value = 7$ **or** $current\ value = 6$ **then**
        $navigate(i, j, k, current\ value)$
      end if
    end for
  end for
end for

Algorithm 6 Navigation recursive algorithm $navigate()$

Input:
Current spacial position with $i = (1, ..., x)$, $j = (1, ..., y)$, $k = (1, ..., z)$.
$level \leftarrow$ Current level energy value of the position.
Output: A set of pairs of voxels which describes the connections.

$current\ voxel \leftarrow getVoxel(i, j, k)$
$neighbors \leftarrow adjacentsVoxelsOf(current\ voxel)$
for all $voxel$ in $neighbors$ **do**
  if $voxel.meetsRamificationCriteria()$ **then**
    $connect(i, j, k, voxel.position())$
    $navigate(voxel.position(), level - 1)$
  end if
end for
{$connect()$ checks that the given pair doesn't belong to the already made connections in both orders, if so connects.}