data = {};  
nmin = 1;

(* Performs the clustering algorithm with tMax steps *)
clustering[tMax_, C1_] := Module[{t, step, idxX},
    (* Init clusters *)
    t = 0;
    C1[t] = RandomSample[data, Length[data]/2];
    Length[C1] = 0;
    C2[t] = Complement[data, C1[t]];

    step[idxCsource_, idxCtarget_] := Module[{fVal},
        fVal = f[x[idxX], C[idxCsource[t]], C[idxCtarget[t]]];
        Length[C[idxCsource[t]]] > nmin;
        If[fVal <= 0,
            (* No improvement in variance, leave point in cluster *)
            C[idxCsource[t + 1]] = C[idxCsource[t]];
            C[idxCtarget[t + 1]] = C[idxCtarget[t]];
            C[idxCsource[t + 1]] = Complement[C[idxCsource[t]], {x[idxX]}];
            C[idxCtarget[t + 1]] = Union[C[idxCtarget[t]], {x[idxX]}];
        ];
    ];

    While[t <= tMax && Length[data] > 2,
        (* Access data points in a ring *)
        idxX = Mod[t, Length[data]] + 1;
        [ step[1, 2] MemberQ[C1[[t]], x[idxX]],
          step[2, 1] MemberQ[C2[[t]], x[idxX]] ];
        t++;
    ];

(* Import common commands *)
functionsFilename = FileNames["*ExchangeClusteringAlgorithm_Functions*", NotebookDirectory[]];
Assert[Length[functionsFilename] = 1];
nb1 = NotebookOpen[functionsFilename[[1]]];
NotebookEvaluate[nb1];

In[15]:= NotebookClose[nb1];
Create own dataset

In[16]:= DynamicModule[{pt = {0, 0}, generateRandom, clear},
generateRandom = (
    data = Union[data, Table[{RandomReal[{0, 10}], RandomReal[{0, 10}]}, {i, 1, 10}]];
) &;
clear = (data = {}) &;
Manipulate[
    data = DeleteDuplicates[data];
    ClickPane[
        Dynamic@Table[
            ListPlot[Append[data, {-1, -1}],
            PlotRange -> {{0, 10.1}, {0, 10.1}},
            ImageSize -> Large,
            AxesLabel -> {"x_1", "x_2"},
            PlotLabel -> "Data points (click to add a new point)"
        ],
        Graphics[{Yellow, Dynamic@Locator[pt]}]
    ], (pt = #; AppendTo[data, pt]) &
    , {action, None, "Action:"},
    Row[{Button["Add random points", generateRandom[]], Button["Clear all", clear[]]}] &
]

Out[16]=

Action: Add random points Clear all

Data points (click to add a new point)

In[17]:= data
Out[17]= {}
In[18] :=

\[ tMax = \text{Length}[\text{data}] \times 2; \]
clustering[tMax, {\text{data}[1], \text{data}[2]}];
Manipulate[plotCluster[\tau], \{\tau, 0, \text{tMax}, 0.25\}]

... Part: Part 1 of {} does not exist.

... Part: Part 2 of {} does not exist.

Out[20] = Null
Exchange clustering example

\begin{verbatim}
In[45]:= data = {{0, 0}, {1, 0}, {1.5, 0.8}, {1.5, 1}};
tMax = Length[data] * 2;
clustering[tMax, {data[[1]], data[[4]]}];
Manipulate[plotCluster[τ], {τ, 0, tMax, 0.25}]
\end{verbatim}

\begin{align*}
D_{\text{var}}(0) &= 2.07 \\
D_{\text{var}}(C_1) + D_{\text{var}}(C_2) &= 0 + 1.59 = 1.59 < 2.07 \\
\Rightarrow & \text{ accept exchange}
\end{align*}
In[25]:= \text{plotCluster}[\theta]

\textbf{t} = 0 \text{ (starting situation)}

Clusters in the beginning

In[26]:= \text{C1}[0]
Out[26]= \{\{0, 0\}, \{1.5, 1\}\}

In[27]:= \text{C2}[0]
Out[27]= \{\{1, 0\}, \{1.5, 0.8\}\}

with the initial centroids

In[28]:= \text{centroid[C1[0]]}
Out[28]= \{0.75, \frac{1}{2}\}

In[29]:= \text{centroid[C2[0]]}
Out[29]= \{1.25, 0.4\}

and the initial variance.

In[30]:= \text{varBefore} = \text{variance[C1[0], C2[0]]}
Out[30]= 2.07

New variance value after the point \text{x}^1 moves from the first to the second cluster.
In[31]:= varAfter = variance[{data[[4]]}, {data[[1]], data[[2]], data[[3]]}]
Out[31]= 1.59333

Ans we see that the change of variance

In[32]:= varBefore - varAfter
Out[32]= 0.476667

is indeed the same as using the update rule.

In[33]:= f[data[[1]], C_1[0], C_2[0]]
Out[33]= 0.476667

And as this value is positive (the variance could be improved), we accept the exchange.
k-means vs. exchange clustering

```plaintext
In[34]:= data = {{1, 2}, {1, 1}, {0.5, 0.5}};
tMax = Length[data] * 2;
clustering[tMax, {data[[1]], data[[2]]}];
Manipulate[plotCluster[τ], {τ, 0, tMax, 0.25}]
```

$t = 0$ (starting situation)
In k-means clustering (Lloyd's algorithm) the initial clustering is also the final clustering since each data point is already assigned to the nearest cluster centre.

\[ D_{\text{var}}(6) = 0.25 \]

In[39]:= \text{plotCluster}[tMax]

\( t = 6 \) (result)

\( \begin{align*}
\text{In}[39]:= & \text{c1 = Mean[data[[1 ;; 2]]]} \\
\text{Out}[39]= & \{1, \frac{3}{2}\} \\
\text{In}[40]:= & \text{c2 = Mean[{data[[3]]}]} \\
\text{Out}[40]= & \{0.5, 0.5\} \\
\text{In}[41]:= & \text{Nearest[{c1, c2}, data[[1]]]} \\
\text{Out}[41]= & \{1, \frac{3}{2}\} \\
\text{In}[42]:= & \text{Nearest[{c1, c2}, data[[2]]]} \\
\text{Out}[42]= & \{1, \frac{3}{2}\} \\
\text{In}[43]:= & \text{Nearest[{c1, c2}, data[[3]]]} \\
\text{Out}[43]= & \{0.5, 0.5\} \\
\end{align*} \)
The crucial part, why the exchange algorithm performs better here, is that in k-means we only re-assign points based on the distance to the cluster centres. But these centres are fixed. In the exchange algorithm, however, the motion of the new cluster centres (due to the exchange) is included as well. This behaviour is exactly what makes the difference in the example here.

The online phase guarantees that we end up in a local minimum of the variance criterion (see Mat lab documentation where they combine both algorithms). If k-means performs a change, the variance criterion decreases as well (this is what the theorem says). However, the theorem says nothing about that we decrease so long until we reach a local minimum. We just don’t increase the variance on the way.

The k-means algorithm has a better computational performance since it works in a batch and not in an online fashion. In the exchange algorithm, we have to re-calculate cluster centres after every performed step whereas in k-means we only need to re-calculate the cluster centres after all points are considered. The performance bottleneck can easily be seen when trying to parallelize the code. In k-means we have basically (pseudocode):

```plaintext
reassign() {
    for each data point
        find the nearest cluster centre
}

recalculateCentres() {
    for each cluster
        calculate the mean all cluster points
}
```
where it is no problem to run both for-loops in parallel since there are no dependencies in between. However, for the exchange algorithm we have

```plaintext
for each data point
  f = varianceChange
  if (f > 0) {
    move point
    re-calculate the cluster centres
  }
```

We cannot run this code in parallel since the check of one point might result in new cluster centres and this, in turn, influences the next checked points. So, we basically execute this algorithm sequentially!