Clear all symbols from previous evaluations to avoid problems

In[190]:=
```
Clear["Global`*"]
```

Clusters on a two-dimensional square lattice

Hoshen-Kopelman algorithm for the square lattice, it returns a 2d array with all sites belonging to a cluster labelled by the same number

In[191]:=
```
HoshenKopelmanSquareLattice[lattice_] :=
  Module[{labelledLattice, n, m, lastLabel, leftLabel, labelAssignments, newLabels},
   {n, m} = Dimensions[lattice];
   labelledLattice = ConstantArray[0, {n, m}];
   labelAssignments = Array[# &, Ceiling[n * m / 2]];
   (* maximum number of clusters *)
   lastLabel = 0;
   Do[ (* loop over the lattice row by row *)
    If[lattice[[i, j]] == 1, (* if site is occupied then ... (otherwise move on) *)
     If[lastLabel == 0,
      (* if this is the first cluster, assign 1 *)
      labelledLattice[[i, j]] = 1;
      lastLabel = 1,
      (* otherwise *)
      If[i == 1,
       (* if we are in the first row,
       just copy the left neighbour (if unoccupied, it will be later corrected) *)
       labelledLattice[[i, j]] = labelledLattice[[i, j - 1]],
       (* otherwise copy first the label of the
        neighbour above (if unoccupied, it will be later corrected) *)
       If[labelledLattice[[i - 1, j]] > 0, labelledLattice[[i, j]] =
         labelAssignments[[labelledLattice[[i - 1, j]]]];
       (* and then for all columns but the first *)
       If[j > 1,
        (* check the left neighbour and possibly resolve conflict *)
        If[labelledLattice[[i, j - 1]] > 0,
         (* if the left site is occupied ... *)
         leftLabel = labelAssignments[[labelledLattice[[i, j - 1]]]];
         (* just to simplify things *)
         If[
          leftLabel < labelledLattice[[i, j]],
          (* if the cluster to the left has a smaller label then the one above,
             resolve conflict and replace all necessary labels *)
          labelAssignments =
           ReplaceAll[labelAssignments, labelledLattice[[i, j]] → leftLabel];
          labelledLattice[[i, j]] = leftLabel,
          (* else *)
          If[
           leftLabel > labelledLattice[[i, j]] && labelledLattice[[i, j]] > 0,
           (* if the cluster to the left has a larger label then the one above,
              resolve conflict the other way and replace all necessary labels *)
           labelAssignments =
            ReplaceAll[labelAssignments, leftLabel -> labelledLattice[[i, j]]],
           (* otherwise just copy the left label *)
           labelledLattice[[i, j]] = leftLabel
```

```
        ]
       ]
      ] (* if the left site is occupied *)
     ](* if we are not in the first column *)
    ]; (* if we are in the first row *)
    If[
     labelledLattice[[i, j]] == 0,
     (* if the label was not
      determined from neighbours then assign a new label *)
     lastLabel++;
     labelledLattice[[i, j]] = lastLabel
    ]
   ](* if it is the first cluster *)
  ], (* if site is occupied *)
  {i, 1, n}, {j, 1, m}
 ];
 (* relabelling *)
 newLabels = Array[# &, Ceiling[n * m / 2]]; (* maximum number of clusters *)
 lastLabel = 1;
 Do[
  If[
   labelledLattice[[i, j]] > 0,
   labelledLattice[[i, j]] = labelAssignments[[labelledLattice[[i, j]]]];
   If[newLabels[[labelledLattice[[i, j]]]] > lastLabel,
    lastLabel++;
    newLabels[[labelledLattice[[i, j]]]] = lastLabel
   ];
   labelledLattice[[i, j]] = newLabels[[labelledLattice[[i, j]]]]
  ],
  {i, 1, n}, {j, 1, m}
 ];
 Return[labelledLattice];
];
```

Function taking the result of the Hoshen-Kopelman algorithm for the square lattice and returning an array with the sizes of clusters

In[192]:=
```
SizesOfClustersSquareLattice[labelledLattice_] :=
 Module[{n, m, lastLabel, nclusters, clusterSizes},
  {n, m} = Dimensions[labelledLattice];
  clusterSizes = ConstantArray[0, Max[labelledLattice]];
  Do[ (* loop over the lattice row by row *)
   If[
    labelledLattice[[i, j]] > 0,
    clusterSizes[[labelledLattice[[i, j]]]]++;
   ],
   {i, 1, n}, {j, 1, m}
  ];
  Return[clusterSizes];
 ];
```

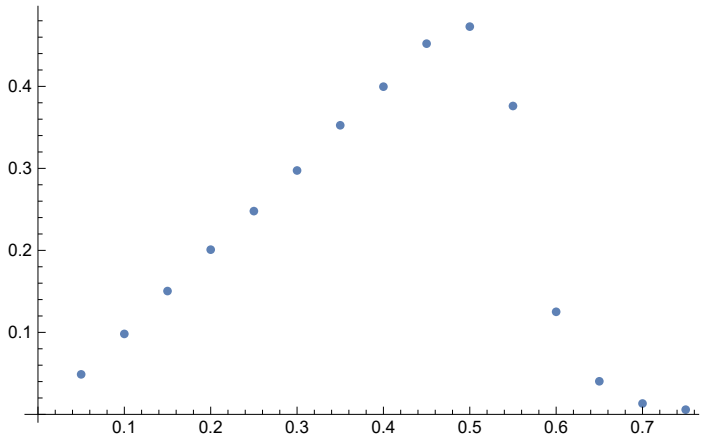
Counting cluster sizes on a square lattice:

In[251]:=
```
{n, m} = {200, 200}; (* lattice size *)
np = 15; (* number of probabilities *)
nconf = 1; (* number of configurations generated for each probability *)
probs = N[0.05 * Range[np]];
clusters = ConstantArray[0.0, {np, n * m}];
Do[ (* loop over probabilities *)
  Do[ (* loop over configurations *)
   Print["Probability p = ", probs[[ip]], ", configuration ", ic];
   lattice = Array[If[RandomReal[] <= probs[[ip]], 1, 0] &, {n, m}];
   sizesOfClusters =
    SizesOfClustersSquareLattice[HoshenKopelmanSquareLattice[lattice]];
   Do[ (* loop over clusters *)
    clusters[[ip, sizesOfClusters[[i]]]]++,
    {i, 1, Length[sizesOfClusters]}
   ],
   {ic, 1, nconf}
  ],
  {ip, 1, np}
 ];
clusters = clusters / (nconf * n * m);
```

```
Probability p = 0.05, configuration 1
Probability p = 0.1, configuration 1
Probability p = 0.15, configuration 1
Probability p = 0.2, configuration 1
Probability p = 0.25, configuration 1
Probability p = 0.3, configuration 1
Probability p = 0.35, configuration 1
Probability p = 0.4, configuration 1
Probability p = 0.45, configuration 1
Probability p = 0.5, configuration 1
Probability p = 0.55, configuration 1
Probability p = 0.6, configuration 1
Probability p = 0.65, configuration 1
Probability p = 0.7, configuration 1
Probability p = 0.75, configuration 1
```

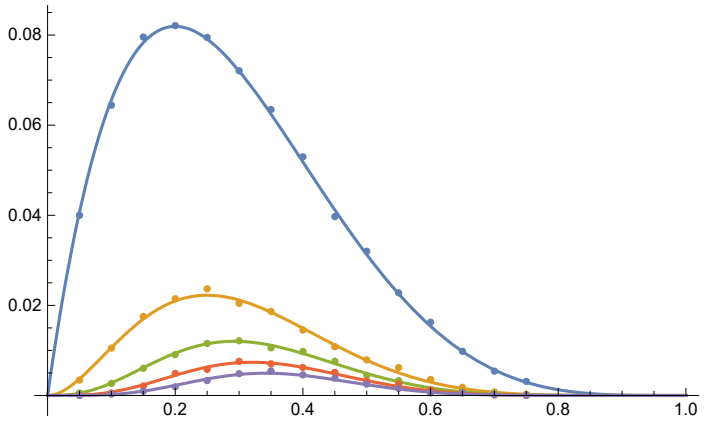
In[255]:=
```
(* checking that sums of n_s(p)*s give approximately p for p < p_c*)
sums = ConstantArray[0.0, np];
Do[ (* loop over probabilities *)
  Do[ (* loop over configurations *)
   sums[[ip]] += clusters[[ip, s]] * s,
   {s, 1, n}
  ],
  {ip, 1, np}
 ];
ListPlot[Transpose[{probs, sums}]]
```

Out[257]=



In[438]:=
```
n1[p_] := p (1 - p)^4;
n2[p_] := 2 p^2 (1 - p)^6;
n3[p_] := p^3 (2 (1 - p)^8 + 4 (1 - p)^7);
n4[p_] := p^4 (2 (1 - p)^10 + 8 (1 - p)^9 + (8 + 1) (1 - p)^8);
n5[p_] := p^5 (2 (1 - p)^12 + (8 + 4) (1 - p)^11 +
      (8 + 8 + 8 + 4) (1 - p)^10 + (8 + 8 + 4) (1 - p)^9 + (1) (1 - p)^8);
pexact = Plot[{n1[p], n2[p], n3[p], n4[p], n5[p]}, {p, 0, 1}, PlotRange → All];
pMC = ListPlot[Table[Transpose[{probs, clusters[[All, i]]}], {i, 1, 5}], PlotRange → All];
Show[pexact, pMC]
```

Out[445]=



Probabilities that a spanning cluster is formed:

In[446]:=
```
np = 41; (* number of probabilities *)
nconf = 500; (* number of configurations generated for each probability *)
latticeSizes = {16, 32, 64};
probs = N[0.5 + 0.005 (Range[np] - 1)];
percProb = ConstantArray[0.0, {Length[latticeSizes], np}];
Do[
  {n, m} = {latticeSizes[[in]], latticeSizes[[in]]}; (* lattice size *)
  Do[ (* loop over probabilities *)
   Print["Lattice size ", n, ", probability = ", probs[[ip]]];
   Do[ (* loop over configurations *)
    lattice = Array[If[RandomReal[] <= probs[[ip]], 1, 0] &, {n, m}];
    labelledLattice = HoshenKopelmanSquareLattice[lattice];
    (* check whether there is a percolating cluster *)
    topClusters = ConstantArray[0, Ceiling[m / 2]];
    Do[ (* loop over the first row *)
     If[
      labelledLattice[[1, j]] > 0,
      topClusters[labelledLattice[[1, j]]] = 1
     ],
     {j, 1, m}
    ];
    percCluster = 0;
    Do[ (* loop over the last row *)
     If[
      labelledLattice[[n, j]] > 0 && labelledLattice[[n, j]] < Ceiling[m / 2],
      If[topClusters[labelledLattice[[n, j]]] == 1, percCluster = 1]
     ],
     {j, 1, m}
    ];
    percProb[[in, ip]] += percCluster,
    {ic, 1, nconf}
   ],
   {ip, 1, np}
  ],
  {in, 1, Length[latticeSizes]}
 ];
percProb = percProb / nconf;
```

In[453]:=
```
ListPlot[Table[Transpose[{probs, percProb[[i]]}], {i, 1, Length[latticeSizes]}],
 PlotRange → All, Joined → True]
```

Out[453]=