Some optimations which are not safe??
Only one up and one down particle taken for potential and kinetic energy and multiplied with 2 (as there are two up and two down which should be symmetrized)

In[•]:=
```
LaunchKernels[]
```
starte Kernels

Out[•]=
```
{KernelObject[ Local KernelID: 1 ], KernelObject[ Local KernelID: 2 ],
 KernelObject[ Local KernelID: 3 ], KernelObject[ Local KernelID: 4 ]}
```

In[•]:=
```
ClearAll[alpha1, alpha2, alpha3, alpha4]
```
lösche alle

```
aaa = {alpha1, alpha2, alpha3, alpha4};
(*Well positions*)
ClearAll[p1, p2, p3, p4];
```
lösche alle

```
wellPositions = {p1, p2, p3, p4};
signWavefunctions = {sn1, sn2, sn3, sn4};
(*Satial wavefunction*)
phiSpatial[x_, center_] :=
  signWavefunctions[[center]] * Exp[-aaa[[center]] * (x - wellPositions[[center]]) ^ 2];
```
Exponentialfunktion

```
(*Spin wavefunction*)
chiSpin[sigma_, spin_] := KroneckerDelta[sigma, spin];
```
Kronecker-Delta

```
(*Combined spatial and spin basis*)
phi[x_, sigma_, center_, spin_] := phiSpatial[x, center] * chiSpin[sigma, spin];
```

In[•]:=

```
(*Generate all valid spin configurations*)
validSpinConfigurations = Permutations[{"up", "up", "down", "down"}];
                          Permutationen
validSpinConfigurations = {{"up", "down", "up", "down"}};

(*Total wavefunction summed over all valid spin configurations*)
 Gesamtsumme
PsiTotal[x_, sigma_] := Sum[Det[Table[phi[x[[i]], sigma[[i]], j, spinConfig[[j]]], {i, 4}, {j, 4}]],
                        ···  D···  Tabelle
   {spinConfig, validSpinConfigurations}];
```

In[•]:=
```
DoInt2[f_] := (Integrate[f, {x3, -Infinity, Infinity}, {x4, -Infinity, Infinity},
               integriere  Unendlic···  Unendlichkeit  Unendlic···  Unendlichkei
   Assumptions → alpha1 > 0 && alpha2 > 0 && alpha3 > 0 && alpha4 > 0 && a > 0])
   Annahmen
```

In[•]:=
```
DoInt3[f_] := (Integrate[f, {x2, -Infinity, Infinity},
               integriere  Unendlic···  Unendlichkeit
   {x3, -Infinity, Infinity}, {x4, -Infinity, Infinity},
        Unendlic···  Unendlichkeit  Unendlic···  Unendlichkeit
   Assumptions → alpha1 > 0 && alpha2 > 0 && alpha3 > 0 && alpha4 > 0 && a > 0])
   Annahmen
```

In[•]:=
```
DoInt4[f_] := (Integrate[Simplify[f], {x1, -Infinity, Infinity},
               integriere  vereinfache  Unendlic···  Unendlichkeit
   {x2, -Infinity, Infinity}, {x3, -Infinity, Infinity}, {x4, -Infinity, Infinity},
        Unendlic···  Unendlichkeit  Unendlic···  Unendlichkeit  Unendlic···  Unendlichkei
   Assumptions → alpha1 > 0 && alpha2 > 0 && alpha3 > 0 && alpha4 > 0 && a > 0])
   Annahmen
```

In[•]:=
```
ClearAll[s1, s2, s3, s4, a, AmpPot]
lösche alle
```

In[•]:=
```
V[x_] := Sum[-Exp[-a (x - xi)^2], {xi, wellPositions}];
         s···  Exponentialfunktion

V[x_] := -AmpPot * Cos[2 * Pi * x * a];
                   Kosi···  Kreiszahl π
```

In[•]:=
```
a
```

Out[•]=
```
a
```

```
In[•]:= KineticEnergy[{x1_, x2_, x3_, x4_}, spins_] :=
          Simplify[-1/2*(D[PsiTotal[{x1, x2, x3, x4}, spins], {x1, 2}] +
          vereinfache  leite ab
            D[PsiTotal[{x1, x2, x3, x4}, spins], {x2, 2}] + D[PsiTotal[{x1, x2, x3, x4}, spins],
            leite ab  leite ab
             {x3, 2}] + D[PsiTotal[{x1, x2, x3, x4}, spins], {x4, 2}])];
                        leite ab
        EnergyIntegrand[{x1_, x2_, x3_, x4_}, spins_] :=
          PsiTotal[{x1, x2, x3, x4}, spins]*(KineticEnergy[{x1, x2, x3, x4}, spins] +
            (V[x1] + V[x2] + V[x3] + V[x4])*PsiTotal[{x1, x2, x3, x4}, spins]);

In[•]:= (*TotalEnergy=ParallelMap[DoInt4,Expand[
                      wende parallel an  multipliziere aus
          EnergyIntegrand[{x1,x2,x3,x4},{"up","down","up","down"}]],Method→"FinestGrained"];*)
                                                                    Methode

In[•]:=
        InputList = List @@ Expand[EnergyIntegrand[{x1, x2, x3, x4}, {"up", "down", "up", "down"}]];
                    Liste  multipliziere aus

In[•]:= (*InputList = InputList[[1;;5]];*)
        Length[InputList]
        Länge
Out[•]= 292

In[•]:= sublength = 40;
        TotalEnergySub[ppp_] := (CloseKernels[];
                                 schließe Kernels
            LaunchKernels[];
            starte Kernels
            ParallelEvaluate[$HistoryLength = 10];
            werte parallel aus  Verlaufslänge
            ParallelMap[DoInt4, ppp, Method → "FinestGrained"]);
            wende parallel an  Methode
        result = Map[TotalEnergySub, Partition[InputList, sublength, sublength, 1, {}]];
                 wende an  partitioniere

In[•]:= result = Flatten[result];
                 ebne ein

In[•]:= Length[result]
        Länge
Out[•]= 292

In[•]:=
        TotalEnergy = Plus @@ result;
                      addiere
        (*memory problems with longer runs in subkernels*)
```

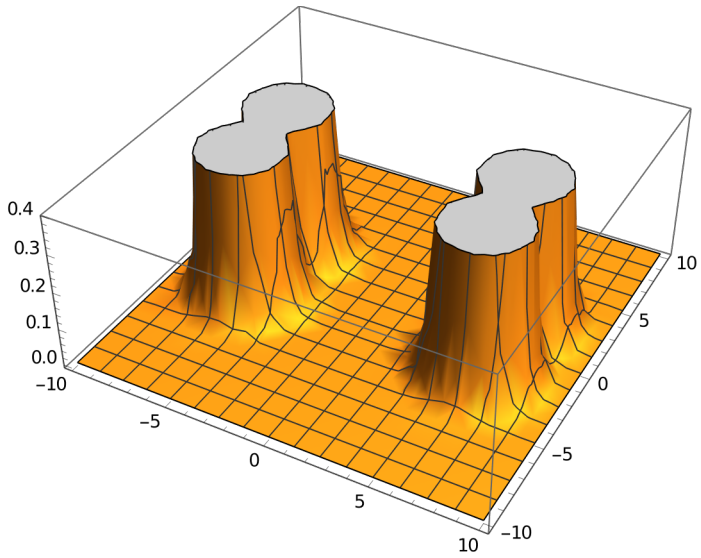
```
In[•]:= CloseKernels[]; LaunchKernels[]; ParallelEvaluate[$HistoryLength = 10];
        schließe Kernels  starte Kernels  werte parallel aus  Verlaufslänge

In[•]:= Normalization = ParallelMap[DoInt4, Expand[
                        wende parallel an  multipliziere aus
          PsiTotal[{x1, x2, x3, x4}, {"up", "down", "up", "down"}]^2], Method → "FinestGrained"];
                                                                    Methode

In[•]:= CloseKernels[]; LaunchKernels[]; ParallelEvaluate[$HistoryLength = 10];
        schließe Kernels  starte Kernels  werte parallel aus  Verlaufslänge

In[•]:= summandCorrelation = ParallelMap[DoInt2,
                             wende parallel an
        Expand[PsiTotal[{x1, x2, x3, x4}, {"up", "down", s3, s4}]^2], Method → "FinestGrained"];
        multipliziere aus                                              Methode

In[•]:= CloseKernels[]; LaunchKernels[]; ParallelEvaluate[$HistoryLength = 10];
        schließe Kernels  starte Kernels  werte parallel aus  Verlaufslänge

In[•]:= TwoParticleCorrelationUpDownududn[xx1_,
          xx2_, al1_, al2_, al3_, al4_, aa_, pp1_, pp2_, pp3_, pp4_] :=
         Sum[summandCorrelation /. {x1 → xx1, x2 → xx2, s3 → sigma34[[1]], s4 → sigma34[[2]],
         summiere
           alpha1 → al1, alpha2 → al2, alpha3 → al3, alpha4 → al4,  a → aa,
           p1 → pp1, p2 → pp2, p3 → pp3, p4 → pp4}, {sigma34, {{"up", "down"}}}];

In[•]:= summandDensityU = ParallelMap[DoInt3,
                          wende parallel an
        Expand[PsiTotal[{x1, x2, x3, x4}, {"up", s2, s3, s4}]^2], Method → "FinestGrained"];
        multipliziere aus                                            Methode

In[•]:= CloseKernels[]; LaunchKernels[]; ParallelEvaluate[$HistoryLength = 10];
        schließe Kernels  starte Kernels  werte parallel aus  Verlaufslänge

In[•]:= OneParticleDensityUududn[xx1_, al1_, al2_, al3_, al4_, aa_, pp1_, pp2_, pp3_, pp4_] :=
         Sum[summandDensityU / Normalization /. {x1 → xx1, s2 → sigma234[[1]], s3 → sigma234[[2]],
         summiere
           s4 → sigma234[[3]], alpha1 → al1, alpha2 → al2, alpha3 → al3, alpha4 → al4,  a → aa,
           p1 → pp1, p2 → pp2, p3 → pp3, p4 → pp4}, {sigma234, {{"down", "up", "down"}}}];

In[•]:= CloseKernels[]; LaunchKernels[]; ParallelEvaluate[$HistoryLength = 10];
        schließe Kernels  starte Kernels  werte parallel aus  Verlaufslänge

In[•]:= summandDensityD = ParallelMap[DoInt3,
                          wende parallel an
        Expand[PsiTotal[{x2, x1, x3, x4}, {s2, "down", s3, s4}]^2], Method → "FinestGrained"];
        multipliziere aus                                            Methode

In[•]:= OneParticleDensityDududn[xx1_, al1_, al2_, al3_, al4_, aa_, pp1_, pp2_, pp3_, pp4_] :=
         Sum[summandDensityD / Normalization /. {x1 → xx1, s2 → sigma234[[1]], s3 → sigma234[[2]],
         summiere
           s4 → sigma234[[3]], alpha1 → al1, alpha2 → al2, alpha3 → al3, alpha4 → al4,  a → aa,
           p1 → pp1, p2 → pp2, p3 → pp3, p4 → pp4}, {sigma234, {{"up", "up", "down"}}}];
```

In[•]:=
```
Energyududn[al1_, al2_, al3_, al4_, aa_, pp1_, pp2_, pp3_, pp4_] :=
 Evaluate[(TotalEnergy / Normalization) /. {alpha1 → al1, alpha2 → al2,
 alpha3 → al3, alpha4 → al4,  a → aa, p1 → pp1, p2 → pp2, p3 → pp3, p4 → pp4}]
```
werte aus

In[•]:=
```
sn1 = sn2 = sn3 = sn4 = 1;
```

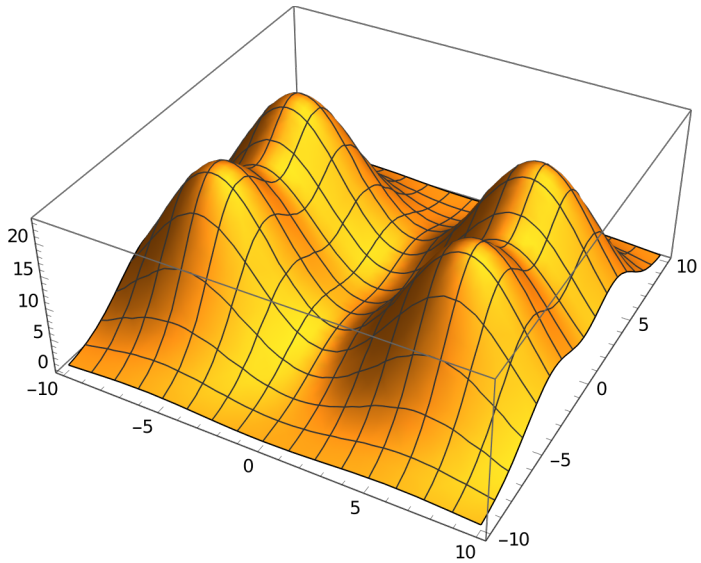
In[•]:=
```
Plot3D[TwoParticleCorrelationUpDownududn[x, y, 0.3, 0.3, 0.3, 0.3, 0.3, -6, -2, 6, 2],
 {x, -10, 10}, {y, -10, 10}]
```
stelle Funktion graphisch in 3D dar

Out[•]=




In[•]:=
```
Plot3D[TwoParticleCorrelationUpDownududn[x, y, 0.03,
 0.03, 0.03, 0.03, 0.03, -6, -2, 6, 2], {x, -10, 10}, {y, -10, 10}]
```
stelle Funktion graphisch in 3D dar

Out[•]=

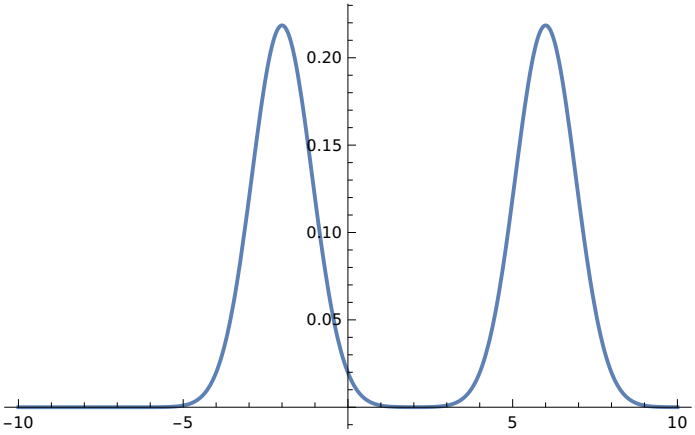



In[•]:=

In[•]:=

In[•]:=
```
Plot[OneParticleDensityDududn[x,
```
stelle Funktion graphisch dar
```
  Sequence @@ {0.3, 0.3, 0.3, 0.3, 0.3, -6, -2, 2, 6}], {x, -10, 10}]
```
Sequenz

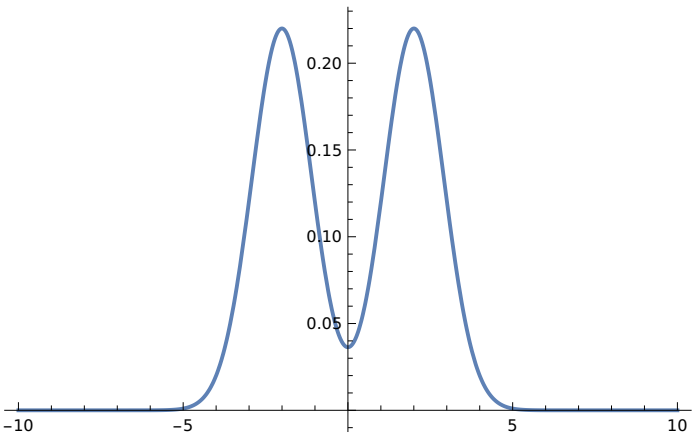
Out[•]=




In[•]:=
```
alpha1 = alpha2 = alpha3 = alpha4 = 0.3
Plot[OneParticleDensityUududn[x,
```
stelle Funktion graphisch dar
```
  Sequence @@ {0.3, 0.3, 0.3, 0.3, 0.3, -2, -2, 2, 6}], {x, -10, 10}]
```
Sequenz

Out[•]= 0.3

Out[•]=




In[•]:=
```
ClearAll[ alpha1, alpha2, alpha3, alpha4, a, p1, p2, p3, p4];
```
lösche alle

In[•]:=
```
ClearAll[ alpha1, alpha2, alpha3, alpha4]
```
lösche alle

```
In[•]:= DoResults[parameters_, ww_] := (
  ppp = {alpha1, alpha2, alpha3, alpha4, Sequence @@ parameters};
                                         Sequenz
  res =
   NMinimize[{Re[Energyududn[alpha1, alpha2, alpha3, alpha4, Sequence @@ parameters]],
    minimiere···  Realteil                                   Sequenz
     alpha1 > 0 && alpha2 > 0 && alpha3 > 0 && alpha4 > 0},
    {alpha1, alpha2, alpha3, alpha4}, Method → "DifferentialEvolution"];
                                       Methode
  ppp = ppp /. res[[2]];
  Print[res];
  gib aus
  ppp1 = Plot[{0.2 * V[x] /. Thread[{a, p1, p2, p3, p4} → parameters],
         stelle Funktion g···  fädle auf
     OneParticleDensityUududn[x, Sequence @@ ppp] +
                                  Sequenz
      OneParticleDensityDududn[x, Sequence @@ ppp], OneParticleDensityUududn[
                                  Sequenz
      x, Sequence @@ ppp], OneParticleDensityDududn[x, Sequence @@ ppp]},
        Sequenz                                      Sequenz
    {x, -ww, ww}, PlotStyle → {Black, Green, {Blue, Dashed}, {Red, Dashed}},
                  Darstellungss··  schw···  grün   blau  gestrichelt  rot  gestrichelt
    PlotLegends → {"0.25*V", "Density", "Density Up", "Density Down"},
    Legenden der Graphik                                   nach unten
    PlotRange → {Automatic, {-0.1, 0.25}}]
    Koordinaten···  automatisch
 )
```

Some tests systematically

```
In[•]:= pw = 1 / 4
        AmpPot = 0.5
Out[•]= 1/4
Out[•]= 0.5
```

4 electrons seperated potentials, up electron plotted, Afterwords the energy per particle of the discussed particles is calculated

In[•]:=
```
DoResults[{pw, 0, -40, 40, 80}, 6]
```

{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}

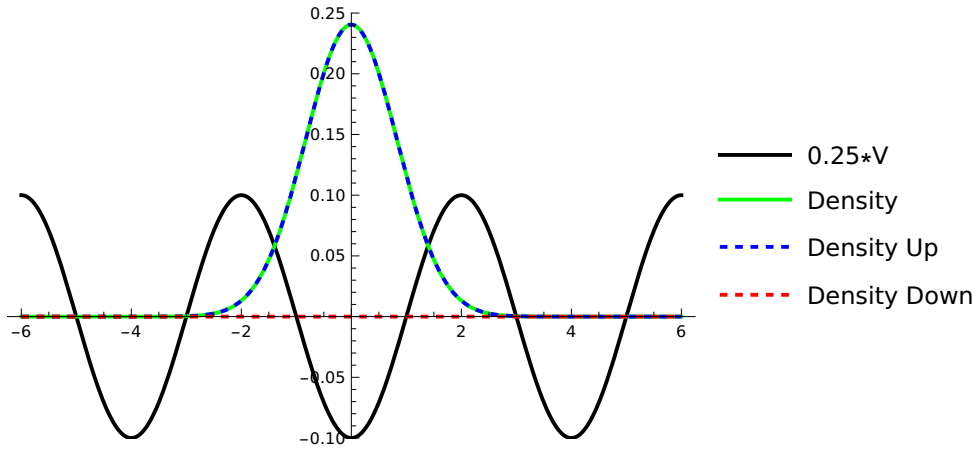
General: Exp[−5811.9] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−5230.71] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−5256.86] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

Out[•]=




In[•]:=
```
eP = res[[1]] / 4
```

Out[•]=
```
-0.032281
```

## Test with other positions

In[•]:=
```
DoResults[{pw, 0, -8, 8, 16}, 20]
```

{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}

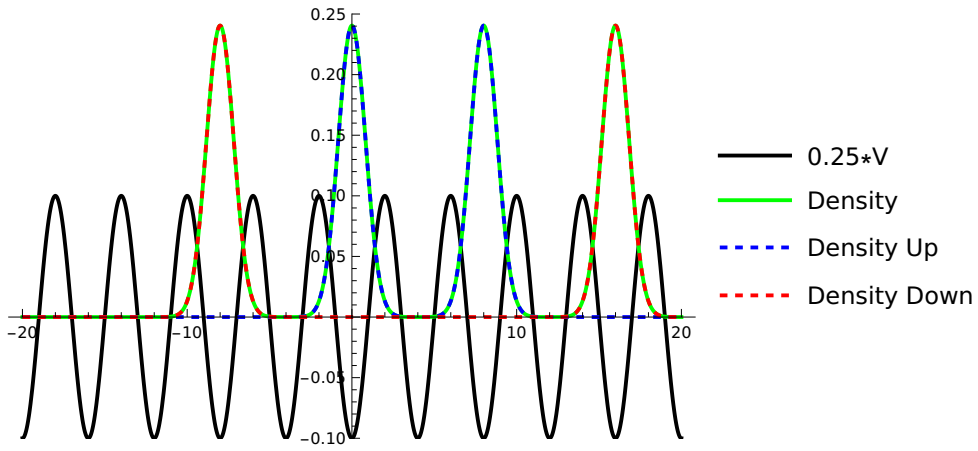
General: Exp[−778.762] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−778.762] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−732.266] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

Out[•]=



In[•]:= **res[[1]] - 3 * eP**

Out[•]= -0.032281

Now two neighboring with oposite spin

In[•]:= **DoResults[{pw, -20, 0, 4, 20}, 7]**

{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}

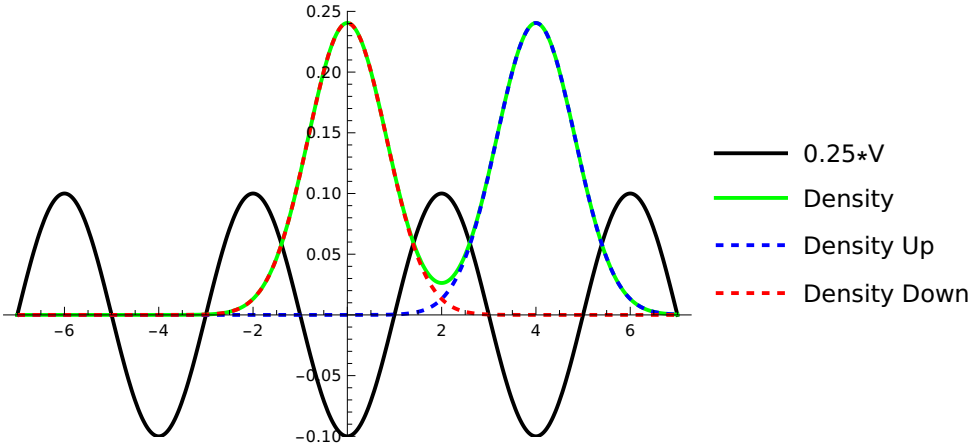
General: Exp[−738.827] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−738.827] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−738.827] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

Out[•]=




In[•]:= **(res[[1]] - 2 * eP) / 2**

Out[•]= -0.032281

Two neighboring with same spin

In[•]:= **DoResults[{pw, -0, -20, 4, 20}, 7]**

{-0.125508, {alpha1 → 0.372887, alpha2 → 0.363244, alpha3 → 0.372887, alpha4 → 0.363244}}

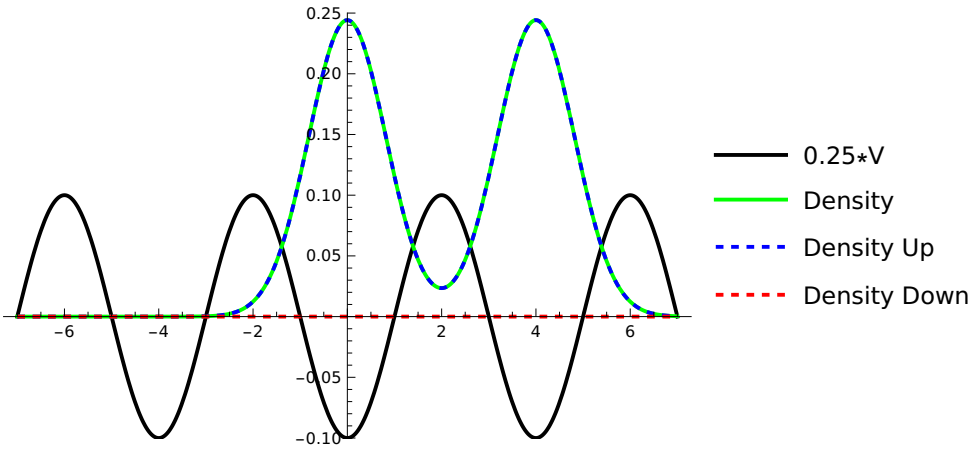
General: Exp[−712.167] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−741.568] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−712.167] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

Out[•]=




In[•]:= **(res[[1]] - 2 * eP) / 2**

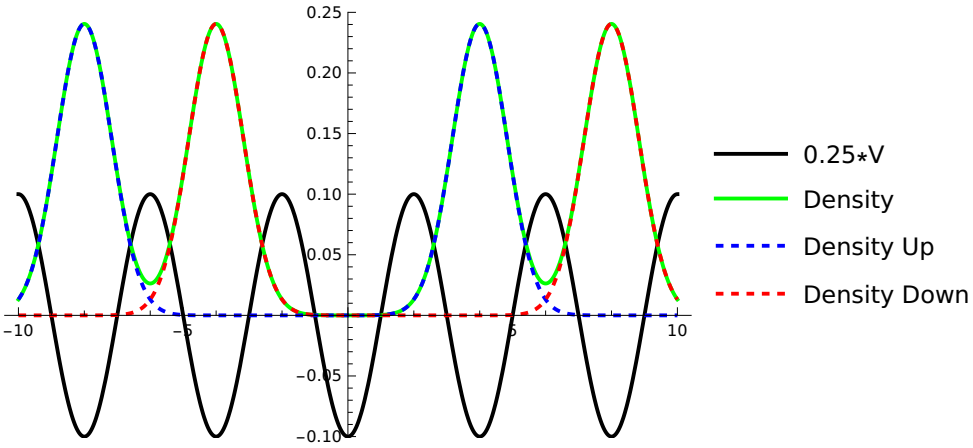
Out[•]= -0.030473

two pairs with oposite spin and an empty inbetween with opposite spin contact

In[•]:= **DoResults[{pw, -8, -4, 4, 8}, 10]**

{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}

Out[•]=




In[•]:= **res[[1]] / 4**

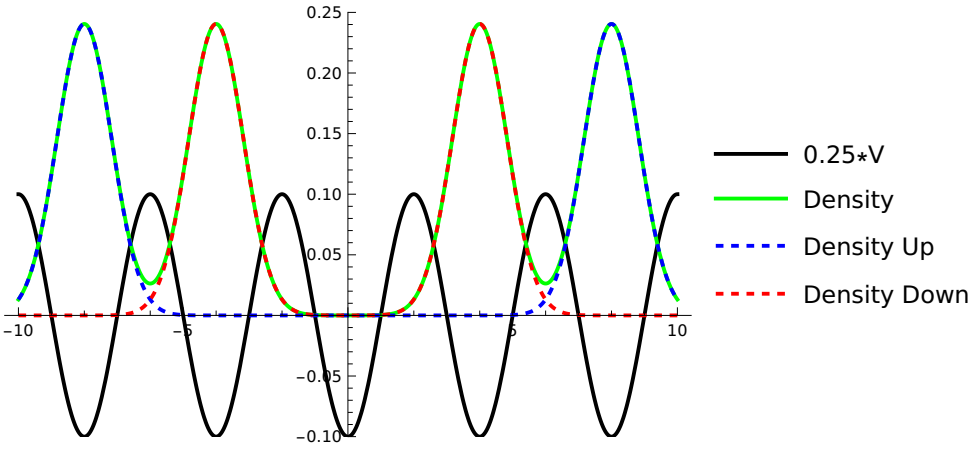
Out[•]= -0.032281

same spin contact

In[•]:=
```
DoResults[{pw, -8, -4, 8, 4}, 10]
```

```
{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}
```

Out[•]=




In[•]:=
```
res[[1]]/4
```

Out[•]=
```
-0.032281
```

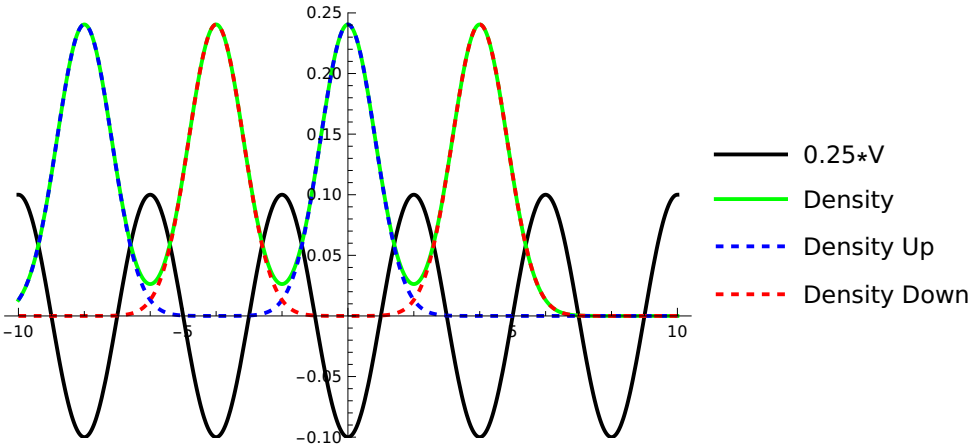
two pairs with oposite spin and without empty inbetween with opposite spin contact

In[•]:=

In[•]:=
```
DoResults[{pw, -8, -4, 0, 4}, 10]
```

```
{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}
```

Out[•]=




In[•]:=
```
res[[1]]/4
```

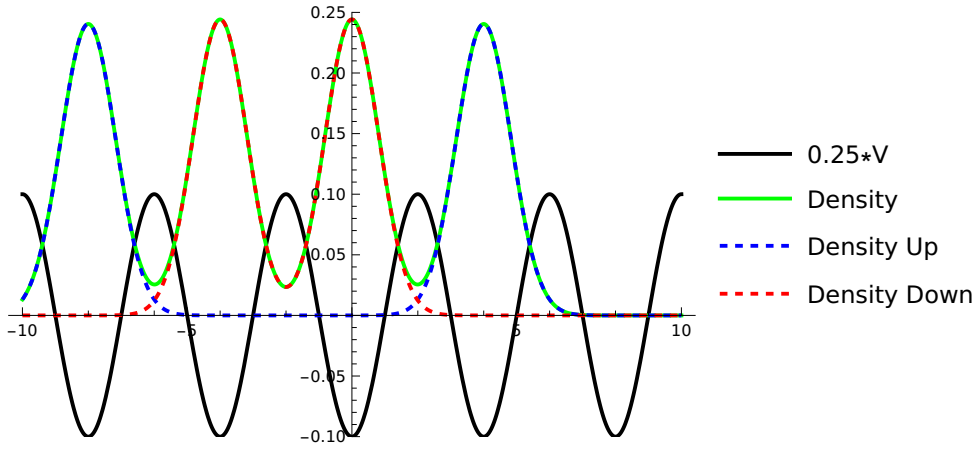
Out[•]=
```
-0.032281
```

two pairs with oposite spin and without empty inbetween with same spin contact

In[•]:=
```
DoResults[{pw, -8, -4, 4, 0}, 10]
```

```
{-0.125508, {alpha1 → 0.363244, alpha2 → 0.372887, alpha3 → 0.363244, alpha4 → 0.372887}}
```

Out[•]=




In[•]:=
```
res[[1]]/4
```

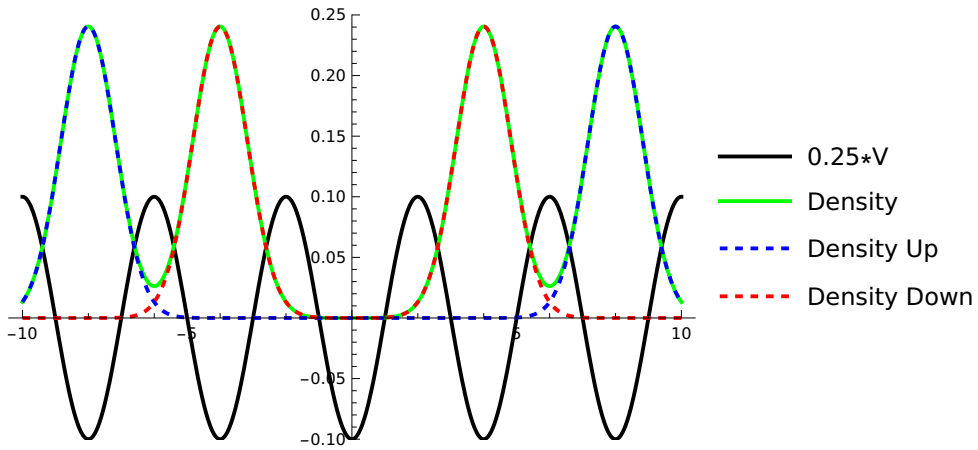
Out[•]=
```
-0.031377
```

two pairs with oposite spin and two empty inbetween with same spin contact

In[•]:=
```
DoResults[{pw, -8, -4, 8, 4}, 10]
```

```
{-0.129124, {alpha1 → 0.363244, alpha2 → 0.363244, alpha3 → 0.363244, alpha4 → 0.363244}}
```

Out[•]=




In[•]:=
```
res[[1]]/4
```

Out[•]=
```
-0.032281
```

sn1 = sn2 = sn3 = sn4 = 1;
sn1 = sn3 = -1;
4 electrons seperated potentials, up electron plotted, Afterwords the energy per particle of the discussed particles is calculated
This gave the same results as above

In[•]:=
```
sn1 = sn2 = sn3 = sn4 = 1;
```

In[•]:=
```
DoResults[parameters_, ww_] := (
  ppp = {alpha1, alpha2, alpha3, alpha4, Sequence @@ parameters};
                                          Sequenz
  res =
   NMinimize[{Re[Energyududn[ alpha1, alpha2, alpha3, alpha4, Sequence @@ parameters]],
    minimiere···  Realteil                                    Sequenz
     alpha1 > 0.1 && alpha2 > 0.1 && alpha3 > 0.1 && alpha4 > 0.1},
    { alpha1, alpha2, alpha3, alpha4}, Method → "DifferentialEvolution"];
                                          Methode
  ppp = ppp /. res[[2]];
  Print[res];
  gib aus
  ppp1 = Plot[{0.2 * V[x] /. Thread[{a, p1, p2, p3, p4} → parameters],
         stelle Funktion g···  fädle auf
     OneParticleDensityUududn[x, Sequence @@ ppp] +
                                          Sequenz
      OneParticleDensityDududn[x, Sequence @@ ppp], OneParticleDensityUududn[
                                          Sequenz
      x, Sequence @@ ppp], OneParticleDensityDududn[x, Sequence @@ ppp]},
        Sequenz                                     Sequenz
    {x, -ww, ww}, PlotStyle → {Black, Green, {Blue, Dashed}, {Red, Dashed}},
                  Darstellungss··  schw···  grün   blau   gestrichelt  rot   gestrichelt
    PlotLegends → {"0.25*V", "Density", "Density Up", "Density Down"},
    Legenden der Graphik                                  nach unten
    PlotRange → {Automatic, {-0.2, 0.5}}]
    Koordinaten···  automatisch
 )
```

In[•]:=
```
pw = 1 / 2
AmpPot = 1.7
```

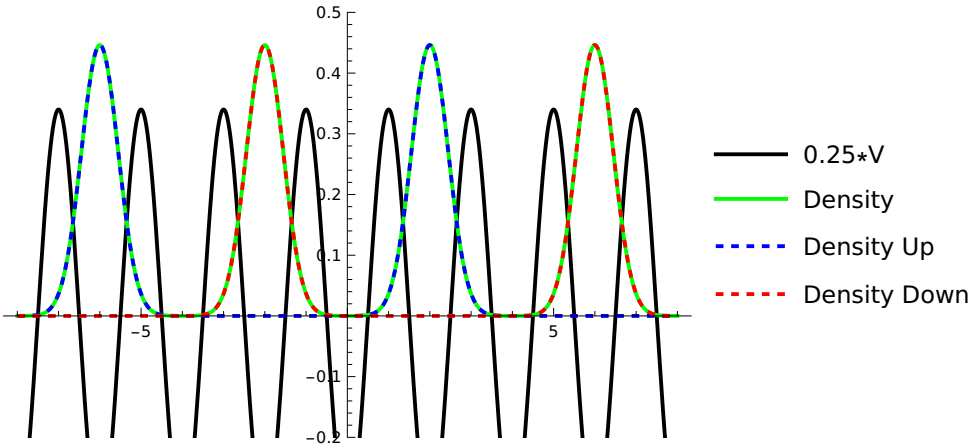
Out[•]= $\frac{1}{2}$

Out[•]= 1.7

In[•]:= **DoResults[{pw, -6, -2, 2, 6}, 8]**

{-0.0344137, {alpha1 → 1.25067, alpha2 → 1.25067, alpha3 → 1.25067, alpha4 → 1.25067}}

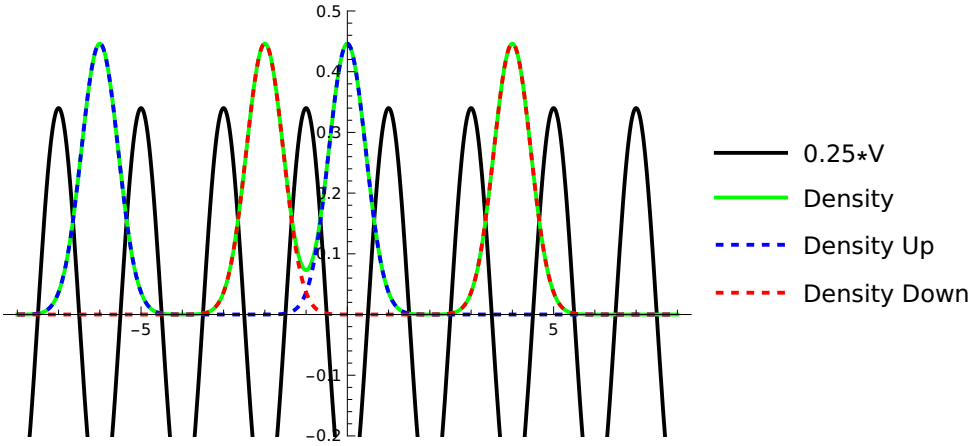
Out[•]=




In[•]:= **DoResults[{pw, -6, -2, 0, 4}, 8]**

{-0.0344137, {alpha1 → 1.25067, alpha2 → 1.25067, alpha3 → 1.25067, alpha4 → 1.25067}}

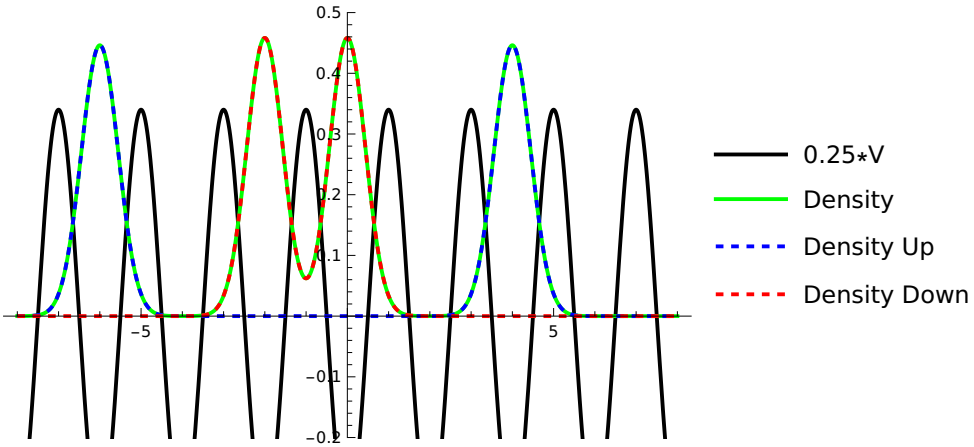
Out[•]=




In[•]:= **DoResults[{pw, -6, -2, 4, 0}, 8]**

{-0.0105665, {alpha1 → 1.25067, alpha2 → 1.30719, alpha3 → 1.25067, alpha4 → 1.30719}}

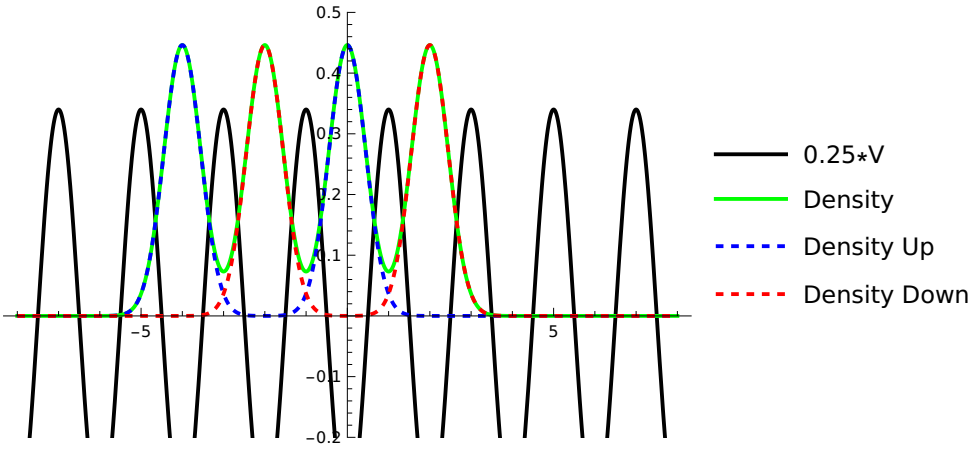
Out[•]=



In[•]:= **DoResults[{pw, -4, -2, 0, 2}, 8]**

{-0.0344136, {alpha1 → 1.25067, alpha2 → 1.25067, alpha3 → 1.25067, alpha4 → 1.25067}}

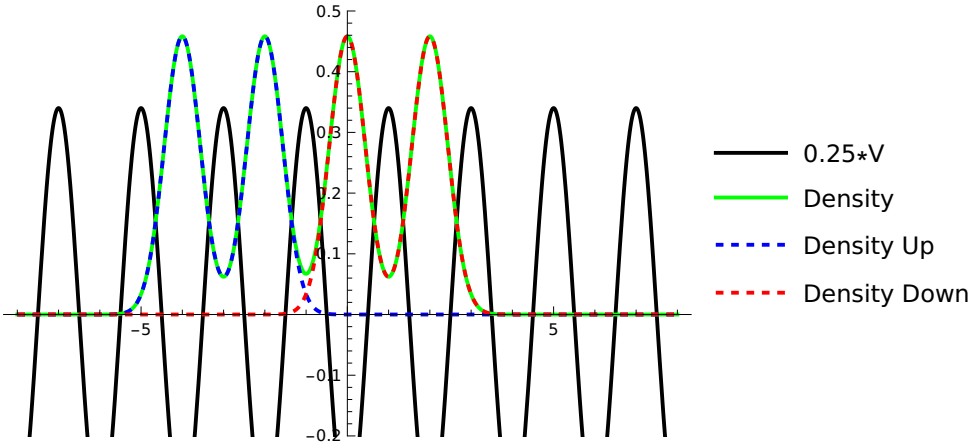
Out[•]=




In[•]:= **DoResults[{pw, -4, 0, -2, 2}, 8]**

{0.0132807, {alpha1 → 1.30719, alpha2 → 1.30719, alpha3 → 1.30719, alpha4 → 1.30719}}

Out[•]=




In[•]:= **DoResults[{pw, -4, -2, 2, 0}, 8]**

{-0.0105665, {alpha1 → 1.25067, alpha2 → 1.30719, alpha3 → 1.25067, alpha4 → 1.30719}}

Out[•]=