

In[1]:=

```
(* :Name: MyMCandMinimization` *)

(* :Author: Karel Houfek *)

BeginPackage["MyMCandMinimization`"];
```

In[2]:=

```
Unprotect[
  MyMinElement,
  MyMinimization,
  MyAmoeba,
  MyAmoebaTry,
  MyRandomNormal,
  MyNRandomNormal,
  MyRandomMetropolis1D,
  MyRandomMetropolisND,
  MyMCIntegration
];

Begin["`Private`"];

Clear[
  MyMinElement,
  MyMinimization,
  MyAmoeba,
  MyAmoebaTry,
  MyRandomNormal,
  MyNRandomNormal,
  MyRandomMetropolis1D,
  MyRandomMetropolisND,
  MyMCIntegration
];

MyMinElement[array_] :=
  Module[{i, imin, min},
    imin = 1;
    min = array[[imin]];
    Do[If[array[[i]] < min, min = array[[i]]; imin = i], {i, 2, Length[array]}];
    Return[imin]
  ];

(* driving function to start Amoeba *)
MyMinimization[guess_, shifts_, f_, tol_] :=
  Module[{n, points, values, i, pointMin, valueMin, niter},
    n = Length[guess];
    (* Initialize points and values for the simplex from guess + shifts *)
    points = ConstantArray[0.0, {n+1, n}];
    values = ConstantArray[0.0, n+1];
    Do[
      points[[i, All]] = guess,
      {i, 1, n+1}
    ];
    values[[1]] = f[guess];
    Do[
      points[[i+1, i]] += shifts[[i]];

```

```

        values[[i+1]] = f[points[[i+1,All]]],
        {i, 1, n}
    ];
    (* Call Amoeba to find the minimum *)
    {pointMin, valueMin, niter} = MyAmoeba[points, values, f, tol];
    Return[{pointMin, valueMin, niter}]
];

(* parameters are updated in MyAmoebaTry *)
(* therefore it is necessary to use attribute HoldAll *)
SetAttributes[MyAmoeba, HoldAll];
MyAmoeba[points_, values_, f_, tol_] :=
Module[{iterMax, eps, iter, n, imin, imax, imax2, ytmp, ytry, rtol, psum},
    iterMax = 200;
    eps = 10-10;
    n = Length[values] - 1;
    iter = 0;
    psum = Total[points, 1];
    While[iter < iterMax,
        (* Find indices of minimum and maximum *)
        imin = MyMinElement[values];
        imax = MyMinElement[-values];
        (* Find index of the second maximum *)
        ytmp = values[[imax]];
        values[[imax]] = values[[imin]];
        imax2 = MyMinElement[-values];
        values[[imax]] = ytmp;
        (* Test of precision, if OK, return the minimum *)
        rtol = 2.0 * Abs[values[[imax]] - values[[imin]]] /
            (Abs[values[[imax]]] + Abs[values[[imin]]] + eps);
        If[rtol < tol,
            Return[{points[[imin, All]], values[[imin]], iter}]
        ];
        (* Try a new point on the other side of the simplex from the maximum *)
        ytry = MyAmoebaTry[points, values, psum, imax, -1.0, f];
        iter += 1;
        (* If a new value is minimum, try to move even further *)
        If[ytry <= values[[imin]],
            ytry = MyAmoebaTry[points, values, psum, imax, 2.0, f];
            iter += 1,
            (* Otherwise, if a new value is large, try to move less *)
            If[ytry >= values[[imax2]],
                ytmp = values[[imax]];
                ytry = MyAmoebaTry[points, values, psum, imax, 0.5, f];
                iter += 1;
                (* If still too large, try to shrink the whole simplex towards the minimum *)
                If[ytry >= ytmp,
                    Do[
                        If[i != imin,
                            points[[i, All]] = 0.5 * (points[[i, All]] + points[[imin, All]]);
                            values[[i]] = f[points[[i, All]]]
                        ],
                        {i, 1, n + 1}
                    ];
                ];
            ];
    ];

```

```

        iter += n;
        psum = Total[points, 1]
    ];
];
];
];
Print["Warning: Number of iterations in Amoeba exceeded ", iterMax];
Return[{points[[imin, All]], values[[imin]], iter}];
];

(* parameters are updated in MyAmoebaTry *)
(* therefore it is necessary to use attribute HoldAll *)
SetAttributes[MyAmoebaTry, HoldAll];
MyAmoebaTry[points_, values_, psum_, imax_, t_, f_] :=
Module[{n, ptry, ytry, t1, t2},
n = Length[values] - 1;
t1 = (1.0 - t) / n;
ptry = t1 * psum + (t - t1) * points[[imax, All]];
ytry = f[ptry];
If[ytry < values[[imax]],
values[[imax]] = ytry;
psum = psum - points[[imax, All]] + ptry;
points[[imax, All]] = ptry;
];
Return[ytry];
];

MyRandomNormal[par_] :=
Module[{out = 0.0},
Do[
out = out + RandomReal[],
{j, 1, 12}
];
out = par[[1]] + par[[2]]*(out - 6.0);
Return[out];
];

MyNRandomNormal[n_, par_] :=
Module[{out = ConstantArray[0.0, {n}]},
Do[
Do[
out[[i]] = out[[i]] + RandomReal[],
{j, 1, 12}
];
out[[i]] = par[[1]] + par[[2]]*(out[[i]] - 6.0),
{i, 1, n}
];
Return[out];
];

MyRandomMetropolis1D[n_,  $\rho$ _, a_, b_, x0_,  $\delta$ _, skip_] :=
Module[{x = x0, xtrial, ratio,  $\rho$ x, out},
out = ConstantArray[0.0, {n}];
 $\rho$ x =  $\rho$ [x];

```

```

Do[
  Do[
    xtrial = x +  $\delta$  * RandomReal[{-1, 1}];
    If[xtrial > a && xtrial < b,
      ratio =  $\rho$ [xtrial] /  $\rho$ x;
      If[ratio >= 1 || ratio > RandomReal[], x = xtrial;  $\rho$ x =  $\rho$ x * ratio]
    ],
    {j, 1, skip}
  ];
  out[[i]] = x,
  {i, 1, n}
];
Return[out]
];

MyRandomMetropolisND[n_, dim_,  $\rho$ _, a_, b_, x0_,  $\delta$ _, skip_] :=
Module[{x = x0, xtrial, new, ratio,  $\rho$ x, out},
  out = ConstantArray[0.0, {n, dim}];
   $\rho$ x =  $\rho$ [x];
  Do[
    Do[
      xtrial = x +  $\delta$  * RandomReal[{-1, 1}, dim];
      new = 1;
      Do[
        If[xtrial[[d]] < a[[d]] || xtrial[[d]] > b[[d]], new = 0],
        {d, 1, dim}
      ];
      If[new == 1,
        ratio =  $\rho$ [xtrial] /  $\rho$ x;
        If[ratio >= 1 || ratio > RandomReal[], x = xtrial;  $\rho$ x =  $\rho$ x * ratio]
      ],
      {j, 1, skip}
    ];
    out[[i]] = x,
    {i, 1, n}
  ];
  Return[out]
];

MyMCIntegration[points_, f_] :=
Module[{n, ip, fp, int1, int2},
  n = Length[points];
  int1 = 0;
  int2 = 0;
  Do[
    fp = f[points[[ip]]];
    int1 = int1 + fp;
    int2 = int2 + fp * fp,
    {ip, 1, n}
  ];
  Return[{int1 / n, Sqrt[(int2 / n - (int1 / n)^2) / n]}];
];

```

```
End[ ];

(* End `Private` Context. *)

SetAttributes[{
  MyMinElement,
  MyMinimization,
  MyAmoeba,
  MyAmoebaTry,
  MyRandomNormal,
  MyNRandomNormal,
  MyRandomMetropolis1D,
  MyRandomMetropolisND,
  MyMCIntegration
},
{ Protected, ReadProtected }
];

EndPackage[ ]; (* End package Context. *)
```