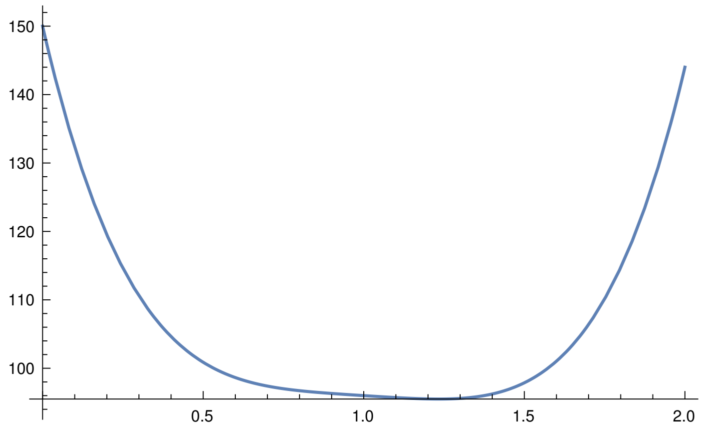
```
In[1]:= (* Initialisation *)
(* Evaluate before starting writing "real code" *)
(* Usage e.g.: "ld [Spacekey]" becomes "⊨",
so writing "a ld 5" turns into "a⊨5" *)
SetOptions[EvaluationNotebook[],
            InputAutoReplacements → {(* special AceGen assignment operators: *)
   "ld" → "⊨", "ls" → "⊢", "rd" → "⫤", "rs" → "⊣",
                                    (* brackets and symbols: *) "dbl" → "⟦",
   "dbr" → "⟧", "lcb" → "{", "rcb" → "}", "lsb" → "[", "rsb" → "]", "->" → "→",
                                    (* shortcuts for
    starting/ending a comment block: *) "co" → "(*", "cc" → "*)"
                                   }
          ]
(* Output the current time,
so we know when AceGen has been executed the last time *)
Now

Out[2]= Fri 14 Jun 2024 13:08:08 GMT+2

In[3]:= (* Clear all old variables initially to have a fresh start *)
ClearAll["Global`*"]
(* Start AceGen *)
<< AceGen`;

NAME = "SMSDo_Loops";
SMSInitialize[NAME, "Language" → "Matlab", "Mode" → "Optimal"];

(* Define the maximum number of Newton-Raphson iterations,
as this also defines the size of convergenceHistory$$ *)
n_iterations = 20;

(* Start a module, which represents the to be created function,
with name "NAME" and the specified input and output arguments *)
SMSModule[NAME, Real[displacement$$, parameter$$,
   convergedValue$$, convergenceHistory$$[2, n_iterations]],
          "Input" → {displacement$$, parameter$$},
          "Output" → {convergedValue$$, convergenceHistory$$}
         ];
displacement ⊨ SMSReal[displacement$$];
parameter ⊨ SMSReal[parameter$$];
```

In[11]:=
```
(* Define some energy function and plot it with an exemplary parameter =1 *)
W_energy[utmp_, parameter_] := (1/2 * 100 * (utmp - parameter)^4 + utmp^2 - 5 utmp + 100);
Plot[W_energy[x, 1], {x, 0, 2}]
```

Out[12]=



In[13]:=
```
(* #1 Works: *)
u_tmp ⫤ displacement;
SMSDo[i_NR, 1, n_iterations, 1, {u_tmp}];
   (* Compute the energy to be minimised using the variable u_tmp *)
    W ⊨ W_energy[u_tmp, parameter];
   (* Find the mininum of the energy W by computing the slope of the
 energy which will be iteratively computed to find the minimum (R=0) *)
    R ⊨ SMSD[W, u_tmp];
   (* Give some output on the progress and export it to convergenceHistory$$ *)
   SMSPrintMessage["iteration i_NR=",
  i_NR, ": W=", W, "; R=", R, "; SMSAbs[R]=", SMSAbs[R]];
   SMSExport[u_tmp, convergenceHistory$$[i_NR, 1]];
   SMSExport[SMSAbs[R], convergenceHistory$$[i_NR, 2]];
   (* Check the residual for convergence *)
    SMSIf[ SMSAbs[R] < 1*^-8 ];
       (* If converged, export the current value of u_tmp *)
        SMSExport[u_tmp, convergedValue$$];
        SMSPrintMessage["converged"];
       (* Leave the SMSDo-Loop *)
        SMSBreak[];
    SMSEndIf[];
    (* If the maximum number of Newton-Raphson iterations is reached,
we output the current value and report a convergence failure *)
    SMSIf[ i_NR ≥ n_iterations ];
        SMSExport[u_tmp, convergedValue$$];
        SMSPrintMessage["failed"];
        SMSBreak[];
    SMSEndIf[];
    (* Compute the derivative and update the value of u_tmp *)
    dRdu ⊨ SMSD[R, u_tmp];
    u_tmp ⊣ u_tmp - 1/dRdu * R;
SMSEndDo[];
```

In[33]:=
```
(* #2 Works: Without auxiliary variable "u_tmp"
    SMSDo[i_NR,1,n_iterations,1,{displacement}];
          (* Compute the energy to be minimised using the variable u_tmp *)
     W ⊨ W_energy[displacement,parameter];
          (* Find the mininum of the energy W by computing the slope of the
  energy which will be iteratively computed to find the minimum (R=0) *)
     R ⊨ SMSD[W,displacement];
          (* Give some output on the
  progress and export it to convergenceHistory$$ *)
          SMSPrintMessage["iteration i_NR=",i_NR,
  ": W=",W,"; R=",R, "; SMSAbs[R]=",SMSAbs[R]];
          SMSExport[displacement,convergenceHistory$$[i_NR,1]];
          SMSExport[SMSAbs[R],convergenceHistory$$[i_NR,2]];
          (* Check the residual for convergence *)
     SMSIf[ SMSAbs[R]<1*^-8 ];
                   (* If converged, export the current value of u_tmp *)
         SMSExport[displacement,convergedValue$$];
         SMSPrintMessage["converged"];
                   (* Leave the SMSDo-Loop *)
         SMSBreak[];
     SMSEndIf[];
           (* If the maximum number of Newton-Raphson iterations is reached,
we output the current value and report a convergence failure *)
           SMSIf[ i_NR ≥n_iterations];
         SMSExport[displacement,convergedValue$$];
         SMSPrintMessage["failed"];
         SMSBreak[];
           SMSEndIf[];
           (* Compute the derivative and update the value of u_tmp *)
     dRdu ⊨ SMSD[R, displacement];
     displacement ⊣ displacement - 1/dRdu * R;
SMSEndDo[];
*)
```

In[34]:=
```
(* #3 Wrong. Value of "displacement" does not change. Use of multi-
 valued variable necessary.
   SMSDo[i_NR,1,n_iterations,1,displacement];
    W ⊨ W_energy[displacement,parameter];
    R ⊨ SMSD[W,displacement];
      (* Give some output on the progress and export it to convergenceHistory$$ *)
        SMSPrintMessage["iteration i_NR=",
 i_NR,": W=",W,"; R=",R, "; SMSAbs[R]=",SMSAbs[R]];
        SMSExport[displacement,convergenceHistory$$[i_NR,1]];
        SMSExport[SMSAbs[R],convergenceHistory$$[i_NR,2]];
        (* Check the residual for convergence *)
    SMSIf[ SMSAbs[R]<1*^-8 ];
       SMSExport[displacement,convergedValue$$];
       SMSPrintMessage["converged"];
       SMSBreak[];
    SMSEndIf[];
         (* If the maximum number of Newton-Raphson iterations is reached,
we output the current value and report a convergence failure *)
         SMSIf[ i_NR ≥n_iterations];
       SMSExport[displacement,convergedValue$$];
       SMSPrintMessage["failed"];
       SMSBreak[];
         SMSEndIf[];
    dRdu ⊨ SMSD[R, displacement];
    displacement ⊢ displacement - 1/dRdu * R; (* "⊢" wrong, use "⊣" instead *)
SMSEndDo[];  *)
```

In[35]:=
```
(* #4 Works. Value of "displacement" is sent out of the loop.
  SMSDo[i_NR,1,n_iterations,1,displacement];
    W ⊨ W_energy[displacement,parameter];
    R ⊨ SMSD[W,displacement];
      (* Give some output on the progress and export it to convergenceHistory$$ *)
        SMSPrintMessage["iteration i_NR=",
 i_NR,": W=",W,"; R=",R, "; SMSAbs[R]=",SMSAbs[R]];
        SMSExport[displacement,convergenceHistory$$[i_NR,1]];
        SMSExport[SMSAbs[R],convergenceHistory$$[i_NR,2]];
        (* Check the residual for convergence *)
    SMSIf[ SMSAbs[R]<1*^-8 ];
        SMSPrintMessage["converged"];
        SMSBreak[];
    SMSEndIf[];
         (* If the maximum number of Newton-Raphson iterations is reached,
we output the current value and report a convergence failure *)
         SMSIf[ i_NR ≥n_iterations];
        SMSPrintMessage["failed"];
        SMSBreak[];
         SMSEndIf[];
    dRdu ⊨ SMSD[R, displacement];
    displacement ⊣ displacement - 1/dRdu * R;
SMSEndDo[displacement]; (* necessary to set "displacement" as out_var *)

SMSExport[displacement,convergedValue$$];
*)
```

In[36]:=
```
(* @todo Add more cases and explanations
 as there are many ways to do it wrong here *)
```

In[37]:=
```
(* Output the time at the end of the execution *)
Now
(* Write output file containing all the
  above defined functions introduced by SMSModule *)
(* Create output file named "NAME", '"LocalAuxiliaryVariables " →
  True' is a command to exclude the AceGen internal array "v" from
   the list of input and output arguments of the created subroutine *)
SMSWrite[NAME, "LocalAuxiliaryVariables " → True];
(* Print the content of the just created
  file on screen (sensible only for small file sizes) *)
FilePrint[StringJoin[NAME, Which[SMSLanguage == "Fortran", ".f",
                                 SMSLanguage == "Matlab", ".m",
                                 SMSLanguage == "C++", ".cpp",
                                 SMSLanguage == "C", ".c"
                                ]
                     ]
          ]
```

Out[37]= Fri 14 Jun 2024 13:08:12 GMT+2

File: SMSDo_Loops .m **Size:** 1677 **Time:** 1

Method	SMSDo_Loops
No.Formulae	13
No.Leafs	136

```
%**************************************************************
%* AceGen    7.505 Linux (16 Aug 22)                         *
%*           Co. J. Korelc  2020           14 Jun 24 13:08:13 *
%**************************************************************
% User     : Full professional version
% Notebook : AceGen-SMSDo
% Evaluation time                 : 1 s     Mode  : Optimal
% Number of formulae              : 13      Method: Automatic
% Subroutine                      : SMSDo_Loops size: 136
% Total size of Mathematica  code : 136 subexpressions
% Total size of Matlab code       : 783 bytes

%*********************** F U N C T I O N **************************
function[convergedValue,convergenceHistory]=SMSDo_Loops(displacement,parameter);
persistent v;
if size(v)<140
  v=zeros(140,'double');
end;
v(3)=displacement;
for i4=1:1:20;
 v(6)=-parameter+v(3);
 v(7)=-5e0+2e0*v(3)+200e0*Power(v(6),3);
 disp(sprintf("\n%s  %f %s %f %s %f %s %f ","iteration  i_NR=",i4,":  W=",100e0-5e0*v(3)+(v(3)*v(
  +50e0*Power(v(6),4),";  R=",v(7),";  SMSAbs[R]=",abs(v(7))));
 convergenceHistory(i4,1)=v(3);
```

```
 convergenceHistory(i4,2)=abs(v(7));
 if(abs(v(7))<0.1e-7)
  convergedValue=v(3);
  disp(sprintf("\n%s  ","converged"));
  break;
 else;
 end;
 if(i4>=20)
  convergedValue=v(3);
  disp(sprintf("\n%s  ","failed"));
  break;
 else;
 end;
 v(3)=v(3)-v(7)/(2e0+600e0*(v(6)*v(6)));
end;


function [x]=SMSKDelta(i,j)

if (i==j) , x=1; else x=0; end;

end

function [x]=SMSDeltaPart(a,i,j,k)

l=round(i/j);

if (mod(i,j) ~= 0 | l>k) , x=0; else x=a(l); end;

end

function [x]=Power(a,b)

x=a^b;

end

function [x]=SMSTernaryOperator(a,b,c)

if (c) , x=a; else x=b; end;

end

end
```