```
In[1]:= Get[FileNameJoin[{NotebookDirectory[],
            "OptimizationToolkit.m"}]];
        Names["OptimizationToolkit`*"]

Out[2]= {FactorExpression, Memoize,
         OptimizeDownValues, $DefaultExcludedForms}
```

Simple usage examples

```
In[3]:= exp1 = ToSphericalCoordinates[{x, y, z}]
```

Out[3]= $\left\{\sqrt{x^2+y^2+z^2},\ \mathrm{ArcTan}\left[z,\sqrt{x^2+y^2}\right],\ \mathrm{ArcTan}[x,y]\right\}$

The default output is in `HoldForm` to prevent evaluating and returning the original expression.

```
In[4]:= FactorExpression[exp1]
```

Out[4]= $\mathrm{Block}\left[\{\$0,\$1\},\ \$0=x^2;\ \$1=y^2;\ \left\{\sqrt{z^2+\$0+\$1},\ \mathrm{ArcTan}\left[z,\sqrt{\$0+\$1}\right],\ \mathrm{ArcTan}[x,y]\right\}\right]$

```
In[5]:= exp2 = FromSphericalCoordinates[{r, θ, ϕ}]
Out[5]= {r Cos[ϕ] Sin[θ], r Sin[θ] Sin[ϕ], r Cos[θ]}

In[6]:= FactorExpression[exp2]
Out[6]= Block[{$2}, $2 = Sin[θ];
          {r $2 Cos[ϕ], r $2 Sin[ϕ], r Cos[θ]}]
```

For large expressions, the factored form generally ends up being much smaller.

```
In[7]:= exp3 = RotationTransform[θ, {xr, yr, zr}][{x, y, z}];
        LeafCount[exp3]
Out[8]= 5563

In[9]:= fexp3 = FactorExpression[exp3];
        LeafCount[fexp3]
Out[10]= 919
```

Automatic optimization

Define a function:

In[11]:=
```
ClearAll[f];
f[r_, t_] := Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]},
  {p, -Pi, Pi, Pi/4}]
f[r_, t_, n_] := Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]},
  {p, -Pi, Pi, Pi/n}]
```

Check the definition:

In[14]:=
```
Definition[f]
```

Out[14]=

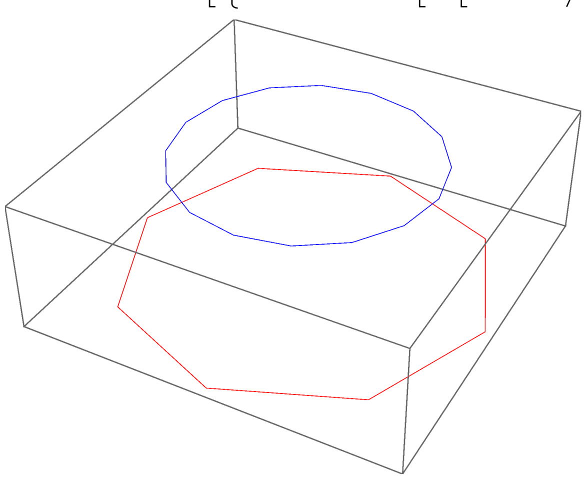
$$f[r\_, t\_] := \text{Table}\left[\{r \cos[p] \sin[t], r \sin[t] \sin[p], r \cos[t]\}, \left\{p, -\pi, \pi, \frac{\pi}{4}\right\}\right]$$

$$f[r\_, t\_, n\_] := \text{Table}\left[\{r \cos[p] \sin[t], r \sin[t] \sin[p], r \cos[t]\}, \left\{p, -\pi, \pi, \frac{\pi}{n}\right\}\right]$$

Example output:

In[15]:=
```
Graphics3D[{Red, Line[f[1, Pi/2]], Blue, Line[f[1, Pi/4, 8]]}]
```

Out[15]=



Use `OptimizeDownValues` to rewrite definitions of functions in an optimized form.

Notice that tables with constant iteration limits are “unrolled”, while those with parametrized limits are not.

In[16]:=
```
Quiet@OptimizeDownValues[f]
```

Out[16]=
```
{HoldPattern[f[r_, t_]] :>
  Block[{$107, $108, $109, $110, $111, $112, $113}, $107 = Sin[t];
   $108 = Cos[t];
   $109 = r $108;
   $110 = 1/Sqrt[2];
   $111 = r $107 $110;
   $112 = {-r $107, 0, $109};
   $113 = r $107;
   {$112, {-$111, -$111, $109}, {0, -$113, $109},
    {$111, -$111, $109}, {$113, 0, $109}, {$111, $111, $109},
    {0, $113, $109}, {-$111, $111, $109}, $112}],
 HoldPattern[f[r_, t_, n_]] :> Block[{$114}, $114 = Sin[t];
   Table[{r Cos[p] $114, r $114 Sin[p], r Cos[t]}, {p, -π, π, π/n}]]}
```

When the option "Rewrite" is False (default), the original definition is unchanged.

In[17]:=
```
Definition[f]
```

Out[17]=
```
f[r_, t_] :=
 Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]}, {p, -π, π, π/4}]

f[r_, t_, n_] :=
 Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]}, {p, -π, π, π/n}]
```

Use “Rewrite” set to True to redefine the function.

```
In[18]:= Quiet@OptimizeDownValues[f, "Rewrite" → True]

Out[18]= {HoldPattern[f[r_, t_]] :→
    Block[{$115, $116, $117, $118, $119, $120, $121}, $115 = Sin[t];
     $116 = Cos[t];
     $117 = r $116;
     $118 = 1/√2;
     $119 = r $115 $118;
     $120 = {-r $115, 0, $117};
     $121 = r $115;
     {$120, {-$119, -$119, $117}, {0, -$121, $117},
      {$119, -$119, $117}, {$121, 0, $117}, {$119, $119, $117},
      {0, $121, $117}, {-$119, $119, $117}, $120}],
   HoldPattern[f[r_, t_, n_]] :→ Block[{$122}, $122 = Sin[t];
     Table[{r Cos[p] $122, r $122 Sin[p], r Cos[t]}, {p, -π, π, π/n}]]}
```

The function now has optimized defintions.

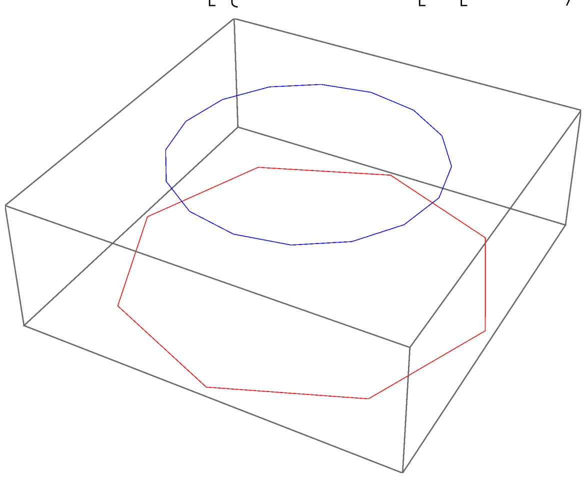
```
In[19]:= Definition[f]

Out[19]= f[r_, t_] :=
  Block[{$115, $116, $117, $118, $119, $120, $121}, $115 = Sin[t];
   $116 = Cos[t];
   $117 = r $116;
   $118 = 1/√2;
   $119 = r $115 $118;
   $120 = {-r $115, 0, $117};
   $121 = r $115;
   {$120, {-$119, -$119, $117}, {0, -$121, $117},
    {$119, -$119, $117}, {$121, 0, $117}, {$119, $119, $117},
    {0, $121, $117}, {-$119, $119, $117}, $120}]

 f[r_, t_, n_] := Block[{$122}, $122 = Sin[t];
   Table[{r Cos[p] $122, r $122 Sin[p], r Cos[t]}, {p, -π, π, π/n}]]

In[20]:= Graphics3D[{Red, Line[f[1, Pi/2]], Blue, Line[f[1, Pi/4, 8]]}]
```

Out[20]=



Other options: Memoization

```
In[21]:= fibonacci[0] = 0;
         fibonacci[1] = 1;
         fibonacci[n_Integer] := fibonacci[n - 1] + fibonacci[n - 2];
In[24]:= AbsoluteTiming[fibonacci[30]]
Out[24]= {2.98351, 832 040}
In[25]:= OptimizeDownValues[fibonacci, "Memoize" → True, "Rewrite" → True];
In[26]:= AbsoluteTiming[fibonacci[30]]
Out[26]= {0.000252267, 832 040}
```

Memoization as a separate function

```
In[27]:= factorial[0] = 1;
         factorial[n : _Integer] := factorial[n - 1] * n;
In[29]:= Memoize[factorial];
In[30]:= Definition[factorial]
Out[30]= factorial[0] := factorial[0] = 1

         factorial[n_Integer] := factorial[n] = factorial[n - 1] n
```

Example of performance gains

Find a unit vector normal to a tetrahedron in 4D space.

```
In[31]:= nexp = Normalize[Det[(e1       e2       e3       e4
                               x2 - x1  y2 - y1  z2 - z1  w2 - w1
                               x3 - x1  y3 - y1  z3 - z1  w3 - w1
                               x4 - x1  y4 - y1  z4 - z1  w4 - w1)]] /.
            {e1 → {1, 0, 0, 0}, e2 → {0, 1, 0, 0}, e3 → {0, 0, 1, 0},
             e4 → {0, 0, 0, 1}}];
In[32]:= sexp = Simplify[nexp, Union[Cases[nexp, _Symbol, Infinity]] ∈
            Reals];
```

In[33]:=
```
tetraNormal[{{x1_, y1_, z1_, w1_}, {x2_, y2_, z2_, w2_},
   {x3_, y3_, z3_, w3_}, {x4_, y4_, z4_, w4_}] := Evaluate[sexp]
```

Find the normals to each face of a pentachoron.

In[34]:=
```
pentachoron = {{x1, y1, z1, w1}, {x2, y2, z2, w2}, {x3, y3, z3, w3},
    {x4, y4, z4, w4}, {x5, y5, z5, w5}};
tetrahedra = Subsets[pentachoron, {4}];
pentachoronFaceNormals = tetraNormal @@@ tetrahedra;
LeafCount[pentachoronFaceNormals]
```

Out[37]=
```
10 586
```

Set up a function that evaluates the original expression.

In[38]:=
```
test1[{{x1_, y1_, z1_, w1_}, {x2_, y2_, z2_, w2_},
    {x3_, y3_, z3_, w3_}, {x4_, y4_, z4_, w4_},
    {x5_, y5_, z5_, w5_}}] := Evaluate[pentachoronFaceNormals]
```

Create an optimized version

In[39]:=
```
test2[{{x1_, y1_, z1_, w1_}, {x2_, y2_, z2_, w2_},
    {x3_, y3_, z3_, w3_}, {x4_, y4_, z4_, w4_},
    {x5_, y5_, z5_, w5_}}] := Evaluate[pentachoronFaceNormals]
Block[{$RecursionLimit = Infinity, $IterationLimit = Infinity},
  OptimizeDownValues[test2, "Rewrite" → True]];
```

The “Output” option allows a compiled function to be returned instead of a Block in HoldForm. This is currently experimental and assumes all symbols are Real.

```
In[41]:= cExp =
  Block[{$RecursionLimit = Infinity, $IterationLimit = Infinity},
   FactorExpression[pentachoronFaceNormals,
    "Output" → CompiledFunction]]

Out[41]= CompiledFunction[
   Argument count: 20
   Argument types: {_Real, _Real, _Real, _Real, _Real, _Real, _Real, _Real, _R
 ]

In[42]:= test3[{{x1_, y1_, z1_, w1_}, {x2_, y2_, z2_, w2_},
     {x3_, y3_, z3_, w3_}, {x4_, y4_, z4_, w4_},
     {x5_, y5_, z5_, w5_}}] :=
  cExp[w1, w2, w3, w4, w5, x1, x2, x3, x4, x5, y1, y2, y3, y4,
   y5, z1, z2, z3, z4, z5]
```

Run some timing tests.

```
In[43]:= testParams = RandomReal[{-1, 1}, {5, 4}];

In[44]:= {t1, res1} = SetPrecision[RepeatedTiming[test1[testParams], 3],
     $MachinePrecision];
  t1

Out[45]= 0.001966930191289383

In[46]:= {t2, res2} = SetPrecision[RepeatedTiming[test2[testParams], 3],
     $MachinePrecision];
  t2

Out[47]= 0.0006830595698472066

In[48]:= {t3, res3} = SetPrecision[RepeatedTiming[test3[testParams], 3],
     $MachinePrecision];
  t3

Out[49]= 0.00001867255958838370
```

Using FactorExpression can yield modest performance gains (about 3x faster in this example). For much larger expressions, the performance increase can get rather ridiculous (several orders of magnitude).

In[50]:=
```
t1/t2
```
Out[50]=
```
2.879588074184166
```
In[51]:=
```
t1/t3
```
Out[51]=
```
105.3380058571627
```

The results are also correct in case you were wondering.

In[52]:=
```
Chop@Total[Abs[Flatten[res1 - res2]]]
```
Out[52]=
```
0
```
In[53]:=
```
Chop@Total[Abs[Flatten[res1 - res3]]]
```
Out[53]=
```
0
```