```
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 wrapped in `Hold` to prevent evaluating and returning the original expression.

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

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

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

Out[5]= $\{r\,\mathrm{Cos}[\phi]\,\mathrm{Sin}[\theta],\ r\,\mathrm{Sin}[\theta]\,\mathrm{Sin}[\phi],\ r\,\mathrm{Cos}[\theta]\}$

```
In[6]:= FactorExpression[exp2]
```

Out[6]= $\mathrm{Hold}\left[\mathrm{Block}\left[\{\$2\},\ \$2=\mathrm{Sin}[\theta];\ \{r\,\$2\,\mathrm{Cos}[\phi],\ r\,\$2\,\mathrm{Sin}[\phi],\ r\,\mathrm{Cos}[\theta]\}\right]\right]$

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_Real, t_Real] :=
 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}]
f[r_, t_] := Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]},
  {p, -Pi, Pi, Pi/4}]
```

Check the definition:

In[15]:=
```
Definition[f]
```

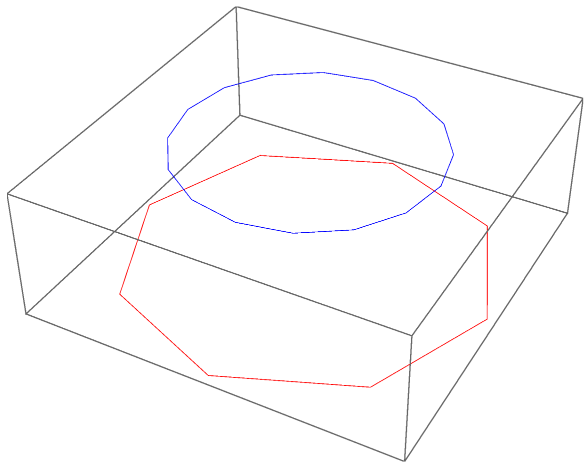
Out[15]=
```
f[r_Real, t_Real] :=
 Table[{r Cos[p] Sin[t], r Sin[t] Sin[p], r Cos[t]}, {p, -π, π, π/4}]

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

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

Example output:

In[16]:=
```
Graphics3D[{Red, Line[f[1, Pi/2]], Blue, Line[f[1, Pi/4, 8]]}]
```

Out[16]=


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

When possible, `OptimizeDownValues` will attempt to compile expressions (feature currently disabled). Notice that tables with constant iteration limits are “unrolled”, while those with parametrized limits are not.

```
In[17]:= Quiet@OptimizeDownValues[f]

Out[17]= {HoldPattern[f[r_Real, t_Real]] :→
    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}]],
   HoldPattern[f[r_, t_]] :→
    Block[{$115, $116, $117, $118, $119, $120, $121}, $115 = Sin[t];
     $116 = Cos[t];
     $117 = r $116;
     $118 = 1/Sqrt[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}]}
```

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

```
In[18]:= Definition[f]

Out[18]= f[r_Real, t_Real] :=
          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}]

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

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

```
In[19]:= Quiet@OptimizeDownValues[f, "Rewrite" → True];
```

The function now has optimized defintions.

In[20]:=
```
Definition[f]
```

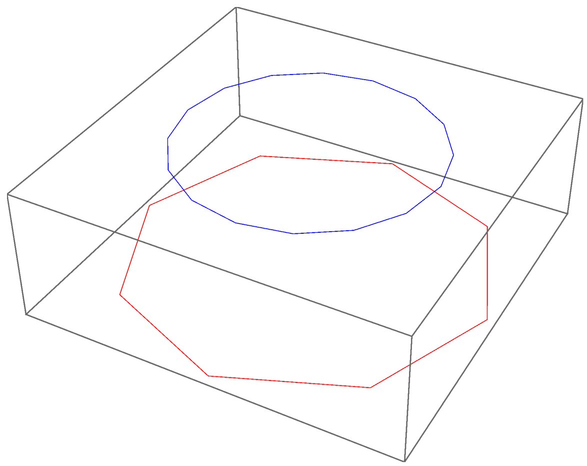
Out[20]=
```
f[r_Real, t_Real] :=
 Block[{$122, $123, $124, $125, $126, $127, $128}, $122 = Sin[t];
  $123 = Cos[t];
  $124 = r $123;
  $125 = 1/Sqrt[2];
  $126 = r $122 $125;
  $127 = {-r $122, 0, $124};
  $128 = r $122;
  {$127, {-$126, -$126, $124}, {0, -$128, $124},
   {$126, -$126, $124}, {$128, 0, $124}, {$126, $126, $124},
   {0, $128, $124}, {-$126, $126, $124}, $127}]

f[r_, t_, n_] := Block[{$129}, $129 = Sin[t];
  Table[{r Cos[p] $129, r $129 Sin[p], r Cos[t]}, {p, -π, π, π/n}]]

f[r_, t_] :=
 Block[{$130, $131, $132, $133, $134, $135, $136}, $130 = Sin[t];
  $131 = Cos[t];
  $132 = r $131;
  $133 = 1/Sqrt[2];
  $134 = r $130 $133;
  $135 = {-r $130, 0, $132};
  $136 = r $130;
  {$135, {-$134, -$134, $132}, {0, -$136, $132},
   {$134, -$134, $132}, {$136, 0, $132}, {$134, $134, $132},
   {0, $136, $132}, {-$134, $134, $132}, $135}]
```

In[21]:=
```
Graphics3D[{Red, Line[f[1, Pi/2]], Blue, Line[f[1, Pi/4, 8]]}]
```

Out[21]=



Other options: Memoization

In[22]:=
```
fibonacci[0] = 0;
fibonacci[1] = 1;
fibonacci[n_Integer] := fibonacci[n - 1] + fibonacci[n - 2];
```

In[25]:=
```
AbsoluteTiming[fibonacci[30]]
```

Out[25]=
```
{3.47007, 832 040}
```

In[26]:=
```
OptimizeDownValues[fibonacci, "Memoize" → True, "Rewrite" → True];
```

In[27]:=
```
AbsoluteTiming[fibonacci[30]]
```

Out[27]=
```
{0.000241022, 832 040}
```

Memoization as a separate function

In[28]:=
```
factorial[0] = 1;
factorial[n : _Integer] := factorial[n - 1] * n;
```

In[30]:=
```
Memoize[factorial];
```

In[31]:=
```
Definition[factorial]
```

Out[31]=
```
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[32]:= 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[33]:= sexp = Simplify[nexp, Union[Cases[nexp, _Symbol, Infinity]] ∈
          Reals];

In[34]:= 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[35]:= 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[38]= 10586
```

Set up a function that evaluates the original expression.

```
In[39]:= 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[40]:= 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[42]:= cExp =
       Block[{$RecursionLimit = Infinity, $IterationLimit = Infinity},
        FactorExpression[pentachoronFaceNormals,
         "Output" → CompiledFunction]]

Out[42]= CompiledFunction[
           Argument count: 20
           Argument types: {_Real, _Real, _Real, _Real, _Real, _Real, _Real, _Real, _Re
         ]

In[43]:= 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[44]:= testParams = RandomReal[{-1, 1}, {5, 4}];

In[45]:= {t1, res1} = SetPrecision[RepeatedTiming[test1[testParams], 3],
        $MachinePrecision];
      t1

Out[46]= 0.002082223536014275
```

```
In[47]:= {t2, res2} = SetPrecision[RepeatedTiming[test2[testParams], 3],
            $MachinePrecision];
         t2
Out[48]= 0.0007238730005432896
In[49]:= {t3, res3} = SetPrecision[RepeatedTiming[test3[testParams], 3],
            $MachinePrecision];
         t3
Out[50]= 0.00001957254828701852
```

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[51]:= t1/t2
Out[51]= 2.876503937087722
In[52]:= t1/t3
Out[52]= 106.3848971263138
```

The results are also correct in case you were wondering.

```
In[53]:= Chop@Total[Abs[Flatten[res1 - res2]]]
Out[53]= 0
In[54]:= Chop@Total[Abs[Flatten[res1 - res3]]]
Out[54]= 0
```