Interval optimisation tutorial

Setup

The `IntervalOptimisation.jl` package can be installed with

```
julia> using Pkg; Pkg.add("IntervalOptimisation")
```

Once the package is installed, it can be imported. Note that you will need also the `IntervalArithmetic.jl` package.

```
using IntervalArithmetic, IntervalOptimisation
```

Function minimisation

The package two main functions are `minimise` and `maximise`. Here we will use `minimise` as example, however `maximise` behaves in an analog way. The main syntax is

```
minimise(f, X, tol=1e-3),
```

where $f : R^n \mapsto R$ is the function to minimize, X is the interval, or interval box, over which we minimize the function. For example, let's consider the function $f(x) = (x-3)^2$ and let us find its global minimum over its whole domain

```
minimise(x->(x-3)^2, -∞..∞)
```

The first value of the results tells us that the minimum value of the function is in the interval $[0, 7.39292e-09]$. The second value tells us that this minimum is achieved in the interval $[2.99964, 3.00019]$. If we want to narrow down this interval, we can use a smaller tolerance

```
minVal, xmin = minimise(x->(x-3)^2, -∞..∞, tol=1e-9)
xmin, [diam(x) for x in xmin]
```

Now the diameter of the minimiser is only 10^{-10} and the minimum is guaranteed to be in that interval.

Multivariate function minimisation

The package can also be used to minimise multivariate functions. Now, the function f will take an array, and X will be an interval box of dimension n. As an example, let us find the minimum of the paraboloid $z = (x-3)^2 + (y-3)^2$ over the box $[-100, 100] \times [-100, 100]$. First let's define the function

```
paraboloid(x) = (x[1]-3)^2 + (x[2]-3)^2
```

and our interval box

```
X = IntervalBox(-100..100, 2)
```

now we can obtain the minimum

```
zmin, xmin = minimise(paraboloid, X, tol=1e-10)
```

the position of the minimum has been found with an accuracy of

```
[diam(x) for x in xmin]
```

Griewank function minimisation

The n-dimensional Griewank function is defined as

$$G_n(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right),$$

for example the 1-dimensional Griewank function is

$$G_1(x) = 1 + \frac{x^2}{4000} - \cos(x)$$

.

and it is commonly used to test optimisation algorithms. This function has the property to have several regularly distributed local minima, but only one global minimum at the origin. Let's define our function

```
G(X) = 1 + sum(abs2, X) / 4000 - prod( cos(X[i] / √i) for i in
1:length(X) )
```

Now let's verify our package finds the minima in several dimensions

```
for N in (1,2, 10, 20, 50)
    res, xmin = minimise(G, IntervalBox(-600..600, N))
    @show N, res, diam(xmin[1])
end
```

Clustering problem

The *clustering problem* is an issue arising in global optimisation with interval arithmetic. Let us consider a simple function

```
f(x) = x^2 - 2x + 1
```

You may recall that interval arithmetic suffers from the *dependency problem*, i.e. if the same variable is repeated in the expression (as in the example above) then evaluating the function will produce an overestimate. Observe

```
f(0..2)
```

using Interval arithmetic we obtain the range $[-3,5]$, while the true range is $[0,1]$. Suppose $x^¿$ is a minimiser for f, then for an ϵ small enough we will have that $f(x^¿) \in f([x^¿+\epsilon, x^¿+2\epsilon])$. Hence, close to the minimiser $x^¿$ we will have intervals not containing $x^¿$ that cannot be thrown away, because they seem to contain $f(x^¿)$. Observe the following example

```
minval, minimisers = minimise(f, 0..2, tol=1e-3);
@show length(minimisers)
```

The function returns 102 minimisers, however only of these contains the true minimiser $x=1$. The following picture illustrates this

```
using Plots
plot(f, 1-0.1, 1+0.1, lw=2)
plot!(IntervalBox.(minimisers, f.(minimisers)), legend=false)
```

Using a stricter tolerance makes things worse, as we will keep bisecting those fake minimisers, obtaining more fake minimisers that cannot be thrown away, Observe

```
minval, minimisers = minimise(f, 0..2, tol=1e-6);
@show length(minimisers)
```

How to solve this problem? A solution is the so called *mean-value form*, instead of computing f(X) with traditional interval arithmetic, we use the following formula

$$f(X) = f(m(X)) + f'(X)(X - m(X))$$

where $m(X)$ denotes the midpoint of X. It can be proved that the true range of $f(X)$ is enclosed into the mean value form. It can also be proved that the mean-value form overestimates reduces the overestimate. Let us define a function to compute the mean-value form, using ForwardDiff to compute the derivative

```
using ForwardDiff

mean_value_form(f, X) = f(mid(X)) + ForwardDiff.derivative(f, X)*(X -
mid(X))
```

Now let us try to minimise the function using the mean-value form

```
minval, minimisers = minimise(X -> mean_value_form(f, X), 0..2,
tol=1e-6);
@show length(minimisers)
@show minimisers
```

As you can see, the number of minimisers has been reduced from 2990 to 3!

This notebook was generated using Literate.jl.