Taylor models tutorial

Setup

The `TaylorModels.jl` package can be installed with

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

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

```
using IntervalArithmetic, TaylorModels
```

Introduction

Given a function f which is continuously derivable $n+1$ times, its nth-order taylor approximation around the point x_0 is given by

$$P_n(x) = \sum_{i=0}^{n} \frac{f^{(n)}(x_0)}{i!}(x - x_0)^i,$$

where $P_n(x)$ is a polynomial of degree n. It is well known that Taylor expansion gives a good approximation in a small neighbourhood around x_0. However it does not give any information on the approximation error.

`TaylorModels.jl` uses interval arithmetic to give a rigorous polynomial approximation of f over the interval $[a, b]$. More formally, the rigorous approximation is given by the pair $(P_n(x), \Delta)$, where

- $P_n(x)$ is a polynomial of degree n, whose coefficients are intervals.
- Δ is the *interval remainder*, and it provides an estimate of all numerical errors (rounding, truncation) encoutered during Taylor expansion computation, i.e. it is guaranteed that $\forall\, x \in [a, b] : |P(x) - f(x)) \in \Delta$.

Taylor models for univariate function

Let us now describe how to get the rigorous Taylor approximation for a function f in an interval a around the point x_0. Let us define our parameters

```
f(x) = x*(x-1.1)*(x+2)*(x+2.2)*(x+2.5)*(x+3)*sin(1.7*x+0.5)
a = -0.5..1
x0 = mid(a) # expansion around the middle point of the domain
```

To create a Taylor model of order n in the interval a around the point x_0 we need the function `TaylorModel1` from `TaylorModels.jl`, (note the 1 stands for 1 dimensional). The signature of this function is Note that `x_0` must be given as an interval, even though it is a single point. Observe the following example, where we create a Taylor approximation of order 6 and 7

```
tm6 = TaylorModel1(6, interval(x0), a)
tm7 = TaylorModel1(7, interval(x0), a)
@show tm6
@show tm7
```

Now to compute the Taylor approximation of our functions we can simply give the previously constructed Taylor model to the function as input

```
ftm6 = f(tm6)
ftm7 = f(tm7)
@show ftm6
@show ftm7
```

The last term of the expression is the interval remainder and the previous ones represent the Taylor polynomial. The result is an object of type `TaylorModel1` with the following attributes

- `dom`: inverval over which the Taylor approximation is computed (a in our example)
- `pol`: polynomial approximation of the function
- `rem`: interval remainder
- `x0`: center of Taylor expansion (x_0 in our example)

Now we can visualize how well our taylor model works using the plot function.

```
using Plots
plot(range(inf(a), stop=sup(a), length=1000), f, lw=2, xaxis="x",
yaxis="f(x)", label="f(x)")
plot!(ftm6, label="6th order")
plot!(ftm7, label="7th order")
```

As you can notice, the Taylor model produces a "band" around the function and the polynomial approximation is guaranteed to be somewhere inside that bar. As you may expect, the higher the order of the model, the narrower the band will be, as the following animation shows

```
orders = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
anim = @animate for n in orders
    tm = TaylorModel1(n, interval(x0), a)
    ftm = f(tm)
    plot(range(inf(a), stop=sup(a), length=1000), f, lw=2, xaxis="x",
yaxis="f(x)",
         label="f(x)", ylims=(-30, 10))
    plot!(ftm, title="$(n)th order")
end
gif(anim, "taylor1_gif.gif", fps = 2)
```

This notebook was generated using Literate.jl.