Lesson 08: Symbolic Math

Most times, when we use MATLAB, we're interested in performing numerical calculations, but what if we wanted to determine analytic solutions to problems? To do this, we can utilize the Symbolic Math Toolbox[1] and turn tedious math homework into a thing of the past![2]

Numerical Stability

We know from MA 111 that we can define the natural logarithm like so:

$$\ln(x) = \int_1^x \frac{1}{t} dt \tag{1}$$

However, $1/t$ can become problematic when dealing with large floating point values. Consider what happens when we try to compute $\ln\left(1 \times 10^{32}\right)$ using this definition numerically:

```matlab
x = log(1e32)

f = @(t) 1 ./ t;
y = integral(f, 1, 1e32)
```

```
x =

   73.6827

y =

   71.6225
```

What we've run into is an issue with the ***numerical stability*** of floating point values. You'll quickly learn that many approaches to solving a problem may result in numerical instability. There are typically a few ways of solving this issue: we can use fixed point arithmetic that gives us the precision we need, use a different numerical approximation, or use an analytic solution and evaluate it to the precision we desire.

[1] `https://www.mathworks.com/products/symbolic.html`
[2] Disclaimer! I am not condoning robbing yourself of valuable practice!

Symbolic Expressions

We can create symbolic variables with `syms`:

```
syms a b c x;

d = a * x^2 + b * x + c
```

```
d =

a*x^2 + b*x + c
```

We can then substitute values into the expression:

```
e = subs(d, [a, b], [4, 3])

% we can also substitute with symbolic expressions!
syms y;
z = subs(d, x, y)
```

```
e =

4*x^2 + 3*x + c

z =

a*y^2 + b*y + c
```

We can also create symbolic functions:

```
syms f(x);

f(x) = 3 * x^2 + 2
```

```
f(x) =

3*x^2 + 2
```

And we can evaluate them:

```
f(4)
```

```
ans =

50
```

Solving and Manipulation

Let's start by solving symbolic expression for a variable:

```
syms x;
solve(x^2 == 4)
```

```
ans =

-2
 2
```

We can also add assumptions about our variables for MATLAB to respect:

```
assume(x >= 0);
solve(x^2 == 4)
```

```
ans =

2
```

We can even solve a system of equations:

```
syms x y;
eqs = [x + y == 4, x - y == 2];
sol = solve(eqs, [x, y])
```

```
sol =

  struct with fields:

    x: 3
    y: 1
```

The `solve()` function is but one of many symbolic solvers available in MATLAB. I encourage you to explore other available solvers to pick the right one for your application: `https://www.mathworks.com/help/symbolic/equation-solving.html`

We can also `simplify()` symbolic expressions:

```
simplify(exp(log(x)))
```

```
ans =

x
```

Note that `simplify()` will try its best to simplify an expression, but it might not always do what you want. Again, MATLAB offers a wide variety of functions to manipulate symbolic expressions, and it is up to you to explore them to pick the right one for your application: `https://www.mathworks.com/help/symbolic/formula-manipulation-and-simplification.html`

Evaluation

Let's go back to our original problem and evaluate it symbolically:

```
syms f(t);
f(t) = 1 / t;
y = int(f, 1, 1e32)
```

```
y =

32*log(10)
```

Notice how MATLAB correctly simplified this integral into an equivalent symbolic expression! And to get a floating point answer, we need to convert it to a `double`:

```
double(y)
```

```
ans =

   73.6827
```

But we can actually evaluate the expression to an arbitrary precision with `vpa()`:

```
vpa(y, 64)
```

```
ans =

73.68272297580946188857572654989965464323524763612073523306649283
```