System Identification – Practical Assignment 8
Output error identification using the Gauss-Newton method

Logistics are as before, see previous labs.

You will develop a function with the exact signature:

```
[index, e1, de1, theta] = oeidentify
```

Each student is assigned an index number in the set 1-8, which needs to be saved to variable `index` at the beginning of the function. The index dictates which data file the student should load. For instance, if you have index 3, you load file `lab8_3.mat`. All these datafiles are already accessible from your function code. Each file contains the identification data in variable `id`, and the validation data in variable `val`. Hint: to increase execution speed and avoid timeouts, save the vectors of inputs and outputs to separate arrays instead of always accessing the `id` object.

From prior knowledge, it is known that the system is first-order, without time delays, and only affected by measurement noise $e(k)$ at the output. This means that the following Output Error form is appropriate to model it:

$$y(k) = \frac{B(q^{-1})}{F(q^{-1})} u(k) + e(k) = \frac{bq^{-1}}{1 + fq^{-1}} u(k) + e(k)$$

with the parameters $\theta = [f, b]^\top$. Our objective will be to implement the prediction error method for this particular model structure, using Gauss-Newton optimization. The algorithm is summarized at the end of this description, in a more direct way than in the lectures and with extra hints, so as to help with implementation. Triangle signs mark comments.

Requirements:

- Compute on paper the recursion formulas for $\varepsilon(k)$, $\frac{d\varepsilon(k)}{d\theta} = [\frac{d\varepsilon(k)}{df}, \frac{d\varepsilon(k)}{db}]^\top$. Hint: see first-order ARMAX case exemplified in the lectures.
- For parameter values $f = b = 1$, apply the formulas obtained to compute the signals $\varepsilon(k)$, $d\varepsilon(k)$ (lines 4 and 6 of the pseudocode). Return the resulting sequences in `e1`, de1, where `de1` is a matrix with two rows and N columns, with $d\varepsilon(k)$ on column k. Note: Grader only checks the first 10 elements of the sequence. Hints: Keep in mind ε is scalar and $d\varepsilon$ is a column vector of two elements. It may also be easier to not represent the signals at 0 explicitly, but code a special case of the update formulas at $k = 1$ instead. The pseudocode already does this.
- Using the recursion formulas above at each iteration, implement the OE identification algorithm, and run it on the identification data. Configure the algorithm as follows: $\theta_1 = [f_1, b_1]^\top = [1, 1]^\top$, $\alpha = 0.5$, $\ell_{\max} = 100$, $\delta = 10^{-4}$. Return the entire sequence of parameter vectors computed in `theta`, a matrix with two rows in which each column ℓ contains θ_ℓ. Note: Grader only checks the first five parameter vectors, and the last one (at convergence). Hints: Make sure you understand the structure of $\frac{dV}{d\theta}$ and $\mathcal{H}$ before you start coding. Use the real inverse (`inv`, not backslash) to implement the update.
- For the near-optimal values of f and b obtained, create an OE model in the system identification toolbox format, using `idpoly`. Note that the syntax of this function is `idpoly(A,B,C,D,F,0,Ts)` where you need to specify the leading zero in B, the leading 1 in F, and the sampling time can be found e.g. in the identification dataset. Use `compare` to see how the model performs on the validation data. Note: Grader checks whether you validated, but you do not need to return the actual model output (since validating θ already took care of model correctness).
- If you still have time, tune α, δ and $\ell_{\max}$ (as well as perhaps θ_1), so as to improve performance.

1: choose stepsize α, initial parameters θ_1, threshold δ, and max iterations $\ell_{\max}$
2: initialize iteration index $\ell = 1$
3: **repeat**
$\triangleright$ lines 4-9 run with the current value of the parameters, θ_ℓ
4: compute directly $\varepsilon(1), d\varepsilon(1)$, using $\varepsilon(0) = 0$, $d\varepsilon(0) = [0, 0]^\top$, $y(0) = 0$, $u(0) = 0$
5: **for** $k = 2, \ldots, N$ **do** $\triangleright$ mind the starting index of the loop!
6: compute $\varepsilon(k)$, $\frac{d\varepsilon(k)}{d\theta}$ with the recursive formulas found $\triangleright$ make sure $\frac{d\varepsilon(k)}{d\theta}$ is a 2x1 vector
7: **end for**
$\triangleright$ make sure $\frac{dV}{d\theta}$ is a 2x1 vector, $\mathcal{H}$ and its inner summation terms are 2x2 matrices
8: compute gradient of the objective function, $\frac{dV}{d\theta} = \frac{2}{N} \sum_{k=1}^{N} \varepsilon(k) \frac{d\varepsilon(k)}{d\theta}$
9: compute approximate Hessian of the objective function, $\mathcal{H} = \frac{2}{N} \sum_{k=1}^{N} \frac{d\varepsilon(k)}{d\theta} \left[\frac{d\varepsilon(k)}{d\theta} \right]^\top$
10: update θ with Gauss-Newton formula: $\theta_{\ell+1} = \theta_\ell - \alpha \mathcal{H}^{-1} \frac{dV}{d\theta}$ $\triangleright$ use `inv`, not $\backslash$
11: increment counter: $\ell = \ell + 1$
12: **until** $\|\theta_\ell - \theta_{\ell-1}\| \leq \delta$, or $\ell > \ell_{\max}$