Motor Physics

Robert Atkinson, Maddy Nguyen, Samantha Hordyk, Cole Welch, and John Fraser
23 July 2018

An exploration of the physics of motors, in both symbolic and numeric form. A basic but sufficiently-complete parameterized model of a motor is presented. The differential equations governing that model are exhibited, and, from those, the (frequency domain) transfer functions of each of the model's variables is developed from first principles. Next, mathematical infrastructure for converting the response of general frequency domain functions input signals is developed, then applied to the transfer functions and their to applied-voltage and applied-torque inputs. This yields time domain functions for each of the model's variables in terms of the (symbolic) value of the model's variables. With those time domain symbolic functions in hand, the steady-state response of the motor as a function of each of the model's parameters is explored.

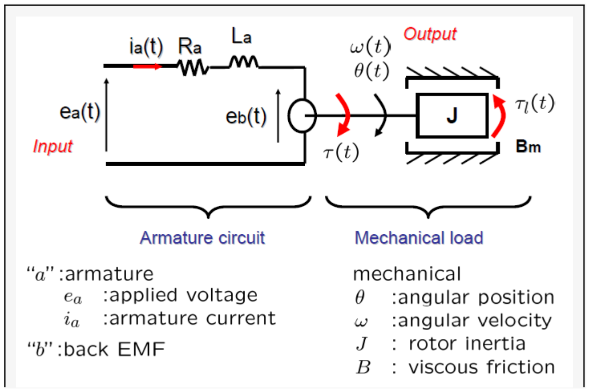
Administrivia

We begin by loading some handy utilities.

```
Get[NotebookDirectory[] <> "Utilities.m"]
```

Motor Model

The following gives a conceptual overview of a DC motor (ref: http://www.egr.msu.edu/classes/me451/jchoi/2008/).



In addition to the the above, we allow here for an externally applied torque.

With that considered, the state of the system can be described by the following variables.

- *ea,* the applied voltage
- *i,* the armature current
- *e,* the back EMF
- *θ*, the angular position
- *τ*, the torque driven by the armature current
- *τ*a, the externally applied torque, if any

The parameter constants that characterize the motor are the following

- J, Moment of inertia
- b, Motor viscous drag constant
- Ke, Electromotive force constant
- Kt, Motor torque constant
- R, Electric resistance

- L Electric inductance

Further, the moment of inertia J and viscous drag constant b can be decomposed into additive contributions from both the motor and the load; we'll return to that later.

```
motorParameterNames = {J, b, Ke, Kt, R, L}

{J, b, Ke, Kt, R, L}
```

Differential Equations

Differential Equation Development

The system can be modelled as a set of differential equations

```
diffEqns = {}

{}
```

Drive torque is proportional to current.

```
diffEqns = Union[diffEqns, {τ[t] == Kt i[t]}]

{τ[t] == Kt i[t]}
```

The back EMF is proportional to velocity.

```
diffEqns = Union[diffEqns, {e[t] == Ke θ'[t]}]

{e[t] == Ke θ′[t], τ[t] == Kt i[t]}
```

Viscous drag reduces the torque, and what's left accelerates the inertial mass. There is an external torque which is linearly dependant angular acceleration as well as having an entirely autonomous term (wherein in many applications is embodied a constant external-torque due to gravity).

```
motorParameterNames = Union[motorParameterNames, {ατ}]
fnτa[t_] := ατ θ''[t] + τa[t]
diffEqns = Union[diffEqns, {τ[t] + fnτa[t] - b θ'[t] == J θ''[t]}]

{b, J, Ke, Kt, L, R, ατ}

{e[t] == Ke θ′[t], τ[t] == Kt i[t], τ[t] + τa[t] - b θ′[t] + ατ θ′′[t] == J θ′′[t]}
```

The voltage drop across the resistive, inductive, and back EMF must equal the applied voltage (due to one of Kirchhoff's laws).

```
diffEqns = Union[diffEqns, {ea[t] == R i[t] + L i'[t] + e[t]}]

{e[t] == Ke θ′[t], ea[t] == e[t] + R i[t] + L i′[t], τ[t] == Kt i[t], τ[t] + τa[t] - b θ′[t] + ατ θ′′[t] == J θ′′[t]}
```

```
diffEqns // prettyPrint
```

$e(t) = \mathrm{Ke}\,\theta'(t)$
$\mathrm{ea}(t) = e(t) + R\,i(t) + L\,i'(t)$
$\tau(t) = \mathrm{Kt}\,i(t)$
$\tau(t) + \tau\mathrm{a}(t) - b\,\theta'(t) + \alpha\tau\,\theta''(t) = J\,\theta''(t)$

Initial Conditions

Some of the initial conditions we can intuit intellectually:

```
initialConditions = {
   θ[0] == 0,
   i[0] == 0,
   e[0] == 0,
   ea[0] == ν,
   τa[0] == T
  };
```

Others we can solve for:

```
others = {i'[0], θ'[0], θ''[0], τ[0]};
(eqnsSubsKnownInitial = diffEqns /. t → 0 /. (initialConditions /. Equal → Rule));
(solvedInitialConditions = (uniqueSolve[eqnsSubsKnownInitial, others] /. Rule → Equal));
(allInitialConditions = solvedInitialConditions ~Join~ initialConditions) // prettyPrint
```

$i'(0) = \frac{v}{L}$
$\theta'(0) = 0$
$\theta''(0) = \frac{\mathrm{T}}{J - \alpha \tau}$
$\tau(0) = 0$
$\theta(0) = 0$
$i(0) = 0$
$e(0) = 0$
$\mathrm{ea}(0) = v$
$\tau\mathrm{a}(0) = \mathrm{T}$

Motor Parameters

We now take a moment to carefully work out the units of each parameter and variable. Some units are obvious and clear. Let’s write those down.

```
siTorqueUnits = parseUnit["N m"] / parseUnit["Radians"];

parameterUnits = {
   R → parseUnit["Ohms"],
   L → parseUnit["Henrys"],
   θ[t] → parseUnit["Radians"],
   θ'[t] → parseUnit["Radians per Second"],
   Ω[t] → parseUnit["Radians per Second"],
   θ''[t] → parseUnit["Radians per Second per Second"],
   α[t] → parseUnit["Radians per Second per Second"],
   i[t] → parseUnit["Amperes"],
   i'[t] → parseUnit["Amperes"] / parseUnit["Second"],
   i''[t] → parseUnit["Amperes"] / parseUnit["Second"] / parseUnit["Second"],
   ea[t] → parseUnit["Volts"],
   e[t] → parseUnit["Volts"],
   τ[t] → siTorqueUnits,
   τa[t] → siTorqueUnits,
   ατ → siTorqueUnits / parseUnit["Radians"] * parseUnit["Seconds"]^2
  } // Association
unitsToQuantities[units_] := Module[{rules, makeQuantity},
  rules = Normal[units];
  makeQuantity = Function[{param, unit}, Quantity[param, unit]];
  (#[[1]] → makeQuantity @@ # &/@ rules) // Association
 ]
parameterQuantities = unitsToQuantities[parameterUnits]
```

$\langle| R \to \text{Ohms}, L \to \text{Henries}, \theta[t] \to \text{Radians}, \theta'[t] \to \frac{\text{Radians}}{\text{Seconds}}, \Omega[t] \to \frac{\text{Radians}}{\text{Seconds}}, \theta''[t] \to \frac{\text{Radians}}{\text{Seconds}^2}, \alpha[t] \to \frac{\text{Radians}}{\text{Seconds}^2}, i[t] \to \text{Amperes}, i'[t] \to \frac{\text{Amperes}}{\text{Seconds}}, i''[t] \to \frac{\text{Amperes}}{\text{Seconds}^2}, ea[t] \to \text{Volts}, e[t] \to \text{Volts}, \tau[t] \to \frac{\text{Meters Newtons}}{\text{Radians}}, \tau a[t] \to \frac{\text{Meters Newtons}}{\text{Radians}}, \alpha\tau \to \frac{\text{Meters Newtons Seconds}^2}{\text{Radians}^2} |\rangle$

$\langle| R \to R\,\Omega, L \to L\,H, \theta[t] \to \theta[t]\,\text{rad}, \theta'[t] \to \theta'[t]\,\text{rad/s}, \Omega[t] \to \Omega[t]\,\text{rad/s}, \theta''[t] \to \theta''[t]\,\text{rad/s}^2, \alpha[t] \to \alpha[t]\,\text{rad/s}^2, i[t] \to i[t]\,A, i'[t] \to i'[t]\,A/s, i''[t] \to i''[t]\,A/s^2, ea[t] \to ea[t]\,V, e[t] \to e[t]\,V, \tau[t] \to \tau[t]\,\text{mN/rad}, \tau a[t] \to \tau a[t]\,\text{mN/rad}, \alpha\tau \to \alpha\tau\,\text{m}\,\text{s}^2\text{N/rad}^2 |\rangle$

A note on units. The Bureau International des Poids et Mesures, the intergovernmental organization that standardizes the SI, officially refers to a radian as “a special name is given to the unit one, in order to facilitate the identification of the quantity involved” (Ref: https://www.bipm.org/en/publications/si-brochure/section2-2-3.html). With that as background, we have here chosen to include radians in all places in which they logically belong, as doing so provides valuable insight regarding the quantities being manipulated. Thus, for example, you will find here torque having units of N m / radian instead of N m, and the like. As, from the SI perspective, this amounts to multiplying or dividing by the unit one, no semantic change is effected.

Aside: that “a special name given to the unit one” is in any way at all different than a first-class “unit” is a perspective reasonable only if one artificially and myopically limits one’s worldview to contain only the seven units whose dimension is necessarily *experimentally* calibrated as opposed to units whose dimension is mathematically or otherwise calibrated, such as radian, becquerel, motor encoder ticks, etc. And the world is chock-full of the latter. But this is a discussion for another time. See also Jacques E. Romain, “Angle as a Fourth Fundamental Quantity”, Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics, Vol. 66B, No. 3, July- September 1962.

Let’s look at how far our tabulation of parameter units takes us:

```
diffEqns /. parameterQuantities // ColumnForm
```

$$e[t]\,V == Ke\left(\theta'[t]\,\text{rad/s}\right)$$
$$ea[t]\,V == (e[t] + R\,i[t] + L\,i'[t])\,A\Omega$$
$$\tau[t]\,\text{mN/rad} == Kt\left(i[t]\,A\right)$$
$$b\left(-\theta'[t]\,\text{rad/s}\right) + (\tau[t] + \tau a[t] + \alpha\tau\,\theta''[t])\,\text{mN/rad} == J\left(\theta''[t]\,\text{rad/s}^2\right)$$

We can see that if we write down the units for either b or J, the remaining units will be fully determined. We choose to specify J.

```
siAngularInertialUnits = parseUnit["kg m^2"] / parseUnit["Radians^2"];

parameterUnits[J] = siAngularInertialUnits;
parameterQuantities = unitsToQuantities[parameterUnits]
diffEqns /. parameterQuantities // ColumnForm
remainingUnits = uniqueSolve[diffEqns[[{1, 3, 4}]], {Ke, Kt, b}]
```

```
⟨|R → R Ω , L → L H , θ[t] → θ[t] rad , θ′[t] → θ′[t] rad/s , Ω[t] → Ω[t] rad/s , θ″[t] → θ″[t] rad/s^2 ,
 α[t] → α[t] rad/s^2 , i[t] → i[t] A , i′[t] → i′[t] A/s , i″[t] → i″[t] A/s^2 , ea[t] → ea[t] V ,
 e[t] → e[t] V , τ[t] → τ[t] m N/rad , τa[t] → τa[t] m N/rad , ατ → ατ m s^2 N/rad^2 , J → J kg m^2/rad^2 |⟩
```

```
e[t] V == Ke ( θ′[t] rad/s )
ea[t] V == (e[t] + R i[t] + L i′[t]) A Ω
τ[t] m N/rad == Kt ( i[t] A )
b ( -θ′[t] rad/s ) + (τ[t] + τa[t] + ατ θ″[t]) m N/rad == J θ″[t] kg m^2/(s^2 rad)
```

$$\left\{Ke \to \frac{e[t]}{\theta'[t]},\ Kt \to \frac{\tau[t]}{i[t]},\ b \to \frac{\tau[t] + \tau a[t] - J\,\theta''[t] + \alpha\tau\,\theta''[t]}{\theta'[t]}\right\}$$

Let’s check how those work.

```
remainingUnits /. parameterQuantities
```

$$\left\{Ke \to \frac{e[t]}{\theta'[t]}\ \mathrm{s\,V/rad},\ Kt \to \frac{\tau[t]}{i[t]}\ \mathrm{m\,N/(A\,rad)},\ b \to \frac{\tau[t] + \tau a[t] - J\,\theta''[t] + \alpha\tau\,\theta''[t]}{\theta'[t]}\ \mathrm{m\,s\,N/rad^2}\right\}$$

Huzzah! They check out (if the units were incompatible, Mathematica would’ve told us). Let’s the remaining units to our kit.

```
(parameterUnits[#] = QuantityUnit[# /. (remainingUnits /. parameterQuantities)]) & /@ remainingUnits[[All, 1]]
parameterQuantities = unitsToQuantities[parameterUnits]
siAngularViscousDragUnits = parseUnit["N m s"] / parseUnit["Radians^2"];
```

$$\left\{\frac{\text{Seconds Volts}}{\text{Radians}},\ \frac{\text{Meters Newtons}}{\text{Amperes Radians}},\ \frac{\text{Meters Newtons Seconds}}{\text{Radians}^2}\right\}$$

```
⟨|R → R Ω , L → L H , θ[t] → θ[t] rad , θ′[t] → θ′[t] rad/s , Ω[t] → Ω[t] rad/s , θ″[t] → θ″[t] rad/s^2 , α[t] → α[t] rad/s^2 ,
 i[t] → i[t] A , i′[t] → i′[t] A/s , i″[t] → i″[t] A/s^2 , ea[t] → ea[t] V , e[t] → e[t] V , τ[t] → τ[t] m N/rad ,
 τa[t] → τa[t] m N/rad , ατ → ατ m s^2 N/rad^2 , J → J kg m^2/rad^2 , Ke → Ke s V/rad , Kt → Kt m N/(A rad) , b → b m s N/rad^2 |⟩
```

Motor Parameters Redux

On Ke & Kt

It is often remarked that, in compatible units, Ke & Kt have the same magnitude and sign. However, while often true, this relation does not always hold. Recall our differential equations:

```
diffEqns // ColumnForm
```

```
e[t] == Ke θ′[t]
ea[t] == e[t] + R i[t] + L i′[t]
τ[t] == Kt i[t]
τ[t] + τa[t] - b θ′[t] + ατ θ″[t] == J θ″[t]
```

In steady state, the current and speed are constant.

```
(ssEqns = diffEqns /. {i'[t] → 0, θ''[t] → 0}) // ColumnForm
(ssEqns = Eliminate[ssEqns, {e[t], b}]) // ColumnForm
```

```
e[t] == Ke θ′[t]
ea[t] == e[t] + R i[t]
τ[t] == Kt i[t]
τ[t] + τa[t] - b θ′[t] == 0
```

```
ea[t] == R i[t] + Ke θ′[t]
τ[t] == Kt i[t]
```

From a power point of view (ref: https://bit.ly/2Lxka0Z), we must have (electrical) power in = (mechanical) power out + (electrical + other) power losses:

```
powerEqns = And @@ {
    pwrIn == ea[t] i[t],
    pwrOut == τ[t] θ'[t],
    pwrLosses == i[t]^2 R + pwrOtherPowerLosses,
    pwrIn == pwrOut + pwrLosses
   };
(ssPowerEqns = ssEqns ~Join~ powerEqns) // prettyPrint
```

$$\mathrm{ea}(t) = R\, i(t) + \mathrm{Ke}\, \theta'(t)$$
$$\tau(t) = \mathrm{Kt}\, i(t)$$
$$\mathrm{pwrIn} = \mathrm{ea}(t)\, i(t)$$
$$\mathrm{pwrOut} = \tau(t)\, \theta'(t)$$
$$\mathrm{pwrLosses} = R\, i(t)^2 + \mathrm{pwrOtherPowerLosses}$$
$$\mathrm{pwrIn} = \mathrm{pwrLosses} + \mathrm{pwrOut}$$

Let’s eliminate a few of the variables that we’re not interested in.

```
kBalanceEqns = Eliminate[ssPowerEqns, {i[t], τ[t], θ'[t], ea[t]}];
kBalanceEqns // prettyPrint
uniqueSolve[kBalanceEqns[[2]], pwrOtherPowerLosses] /. Rule → Equal
```

$$\mathrm{pwrIn} = \mathrm{pwrLosses} + \mathrm{pwrOut}$$
$$\mathrm{Kt}\, \mathrm{pwrOtherPowerLosses} = (\mathrm{Ke} - \mathrm{Kt})\, \mathrm{pwrOut}$$

$$\left\{\mathrm{pwrOtherPowerLosses} == \frac{(\mathrm{Ke} - \mathrm{Kt})\ \mathrm{pwrOut}}{\mathrm{Kt}}\right\}$$

We can conclude that if other power losses are zero, then Ke must equal Kt.

Experimental Motor Data

We have experimental data from several motors (Nguyen, Hordyk, & Fraser, 2017). We load in same.

```
(motorData = Import[NotebookDirectory[] <> "Characterized Motors.xlsx", {"Data", 1}]) // TableForm
```

```
Name         J           b           K        R       L (uH)     L (H)       J
AM 20 A      9.0110-6    0.0022      0.351    2.3     691.       0.000691    9.011×10^-6
AM 20 B      9.0110-6    0.0025      0.389    1.9     684.       0.000684    9.011×10^-6
AM 20 C      8.9310-6    0.0028      0.385    5.1     717.       0.000717    8.931×10^-6
AM 40 A      2.2210-5    0.2269      0.753    2.5     674.       0.000674    0.00002221
AM 40 B      1.7410-5    0.56        0.705    3.8     705.       0.000705    0.00001741
AM 40 C      2.4710-5    0.018       0.763    2.1     716.       0.000716    0.00002471
AM 60 A      1.0410-5    0.033       1.066    3.3     694.       0.000694    0.00001041
AM 60 B      8.4210-6    0.02        1.076    5.1     696.       0.000696    8.421×10^-6
AM 3.7 A     2.7910-5    0.00014     0.099    8.9     679.       0.000679    0.00002791
AM 3.7 B     3.1510-5    0.000176    0.108    2.6     797.       0.000797    0.00003151
AM 3.7 C     3.0910-5    0.00017     0.105    8.7     880.       0.00088     0.00003091
Matrix A     9.4310-6    0.00151     0.34     3.8     718.       0.000718    9.431×10^-6
Matrix B     7.7610-6    0.00191     0.363    7.8     777.       0.000777    7.761×10^-6
Matrix C     7.2310-6    0.00186     0.338    20.6    658.       0.000658    7.231×10^-6
CoreHex A    7.3310-4    0.0112      0.822    3.6     1356.      0.001356    0.0007331
CoreHex B    6.5510-4    0.008       0.858    11.3    1352.      0.001352    0.0006551
CoreHex C    4.5410-4    0.0078      0.711    5.6     1342.      0.001342    0.0004541
```

We build a function to retrieve data from the table. We convert from floating point to rational in order to help delay floating point collapse later on down the line. We allow for optional load inertia and viscous drag.

```
siAngularInertialUnits = parseUnit["kg m^2"] / parseUnit["Radians^2"];
siAngularViscousDragUnits = parseUnit["N m s"] / parseUnit["Radians^2"];
siTorqueUnits = parseUnit["N m"] / parseUnit["Radians"];

Clear[motorParameters]
Options[motorParameters] = {
   JLoad → Quantity[0, siAngularInertialUnits],
   bLoad → Quantity[0, siAngularViscousDragUnits]
  };
motorParameters[motorName_, opts : OptionsPattern[]] := Module[{row, paramValues, quantify, assoc},
   row = Select[motorData, #[[1]] == motorName &][[1]];
   paramValues = #[[1]] → toRational[row[[#[[2]]]]] & /@ {{J, 8}, {b, 3}, {Ke, 4}, {Kt, 4}, {R, 5}, {L, 7}};
   quantify = Function[{name, value},
     name → ((name /. parameterQuantities) /. name → value)
    ];
   assoc = quantify @@ # & /@ paramValues // Association;
   assoc[J] = assoc[J] + OptionValue[JLoad];
   assoc[b] = assoc[b] + OptionValue[bLoad];
   assoc
  ];

motorParameters["AM 60 A", JLoad → Quantity[1, siAngularInertialUnits]]
```

```
       100 001 041                    33                   533              533                    33           347
⟨|J → ─────────── kg m²/rad² , b → ──── m s N/rad² , Ke → ─── s V/rad , Kt → ─── m N/(A rad) , R → ── Ω , L → ─────── H |⟩
       100 000 000                   1000                  500              500                    10         500 000
```

Working Examples

Example motors

We define a motor with a load that we'll use to illustrate examples. We also define a generic, abstract motor that can helps explore things symbolically

```
siAngularInertialUnits
aMotor = motorParameters["AM 60 A", JLoad → Quantity[1, siAngularInertialUnits]]
```

```
Kilograms Meters²
─────────────────
     Radians²
```

```
       100 001 041                    33                   533              533                    33           347
⟨|J → ─────────── kg m²/rad² , b → ──── m s N/rad² , Ke → ─── s V/rad , Kt → ─── m N/(A rad) , R → ── Ω , L → ─────── H |⟩
       100 000 000                   1000                  500              500                    10         500 000
```

```
genericMotor = # → (# /. parameterQuantities) & /@ motorParameterNames
```

```
{b → b m s N/rad² , J → J kg m²/rad² , Ke → Ke s V/rad , Kt → Kt m N/(A rad) , L → L H , R → R Ω , ατ → ατ m s² N/rad² }
```

Laplace Transforms and Transfer Function Models

Laplace Transforms

Returning to the differential equations, we form the Laplace Transform of each, then solve the for the various transforms. First, we apply the LaplaceTransform[] function to each equation. It automatically makes use of the linearity of the transform, and insinuates itself just around the time dependent parts (ie: the parts dependent on the time variable we told it about, namely *t*).

```
leqns = LaplaceTransform[#, t, s] &/@ diffEqns
leqns // prettyPrint
```

```
{LaplaceTransform[e[t], t, s] == Ke (s LaplaceTransform[θ[t], t, s] - θ[0]), LaplaceTransform[ea[t], t, s] ==
  LaplaceTransform[e[t], t, s] + R LaplaceTransform[i[t], t, s] + L (-i[0] + s LaplaceTransform[i[t], t, s]),
 LaplaceTransform[τ[t], t, s] == Kt LaplaceTransform[i[t], t, s],
 LaplaceTransform[τ[t], t, s] + LaplaceTransform[τa[t], t, s] - b (s LaplaceTransform[θ[t], t, s] - θ[0]) +
   ατ (s² LaplaceTransform[θ[t], t, s] - s θ[0] - θ′[0]) == J (s² LaplaceTransform[θ[t], t, s] - s θ[0] - θ′[0])}
```

$$\mathcal{L}_t[e(t)]\,(s) = \mathrm{Ke}\,(s\,(\mathcal{L}_t[\theta(t)]\,(s)) - \theta(0))$$
$$\mathcal{L}_t[\mathrm{ea}(t)]\,(s) = \mathcal{L}_t[e(t)]\,(s) + R\,(\mathcal{L}_t[i(t)]\,(s)) + L\,(s\,(\mathcal{L}_t[i(t)]\,(s)) - i(0))$$
$$\mathcal{L}_t[\tau(t)]\,(s) = \mathrm{Kt}\,(\mathcal{L}_t[i(t)]\,(s))$$
$$\mathcal{L}_t[\tau(t)]\,(s) + \mathcal{L}_t[\tau\mathrm{a}(t)]\,(s) - b\,(s\,(\mathcal{L}_t[\theta(t)]\,(s)) - \theta(0)) + \alpha\tau\left((\mathcal{L}_t[\theta(t)]\,(s))\,s^2 - \theta(0)\,s - \theta'(0)\right) = J\left((\mathcal{L}_t[\theta(t)]\,(s))\,s^2 - \theta(0)\,s - \theta'(0)\right)$$

Next, we walk those equations, picking up those insinuations and sowing them to the wind, reaping them on the outside, and finally removing duplicates

```
allXforms = Reap[Scan[(If[MatchQ[#, _LaplaceTransform], Sow[#]]) &, leqns, Infinity]][[2, 1]] // Union
```

```
{LaplaceTransform[e[t], t, s], LaplaceTransform[ea[t], t, s], LaplaceTransform[i[t], t, s],
 LaplaceTransform[θ[t], t, s], LaplaceTransform[τ[t], t, s], LaplaceTransform[τa[t], t, s]}
```

The voltage transform is input; all the others are outputs. Solve for the outputs. Finally substitute what we know about initial conditions, and simplify as much as we can.

```
voltageXform = LaplaceTransform[ea[t], t, s];
τinXform = LaplaceTransform[τa[t], t, s]
inputXforms = {voltageXform, τinXform};
outputXorms = Complement[allXforms, inputXforms]
rawSolvedXforms = uniqueSolve[leqns, outputXorms]
rawSolvedXforms = rawSolvedXforms /. (allInitialConditions /. Equal → Rule) // FullSimplify
```

```
LaplaceTransform[τa[t], t, s]
```

```
{LaplaceTransform[e[t], t, s], LaplaceTransform[i[t], t, s], LaplaceTransform[θ[t], t, s], LaplaceTransform[τ[t], t, s]}
```

```
{LaplaceTransform[e[t], t, s] →
  -((-Ke Kt L i[0] - Ke Kt LaplaceTransform[ea[t], t, s] - Ke R LaplaceTransform[τa[t], t, s] - Ke L s LaplaceTransform[τa[t], t, s] -
       J Ke R θ′[0] - J Ke L s θ′[0] + Ke R ατ θ′[0] + Ke L s ατ θ′[0]) / (Ke Kt + b R + b L s + J R s + J L s² - R s ατ - L s² ατ)),
 LaplaceTransform[i[t], t, s] → -((-b L i[0] - J L s i[0] + L s ατ i[0] - b LaplaceTransform[ea[t], t, s] - J s
          LaplaceTransform[ea[t], t, s] + s ατ LaplaceTransform[ea[t], t, s] + Ke LaplaceTransform[τa[t], t, s] + J Ke θ′[0] - Ke ατ θ′[0]) /
      (Ke Kt + b R + b L s + J R s + J L s² - R s ατ - L s² ατ)), LaplaceTransform[θ[t], t, s] →
  -((-Kt L i[0] - Kt LaplaceTransform[ea[t], t, s] - R LaplaceTransform[τa[t], t, s] - L s LaplaceTransform[τa[t], t, s] -
       Ke Kt θ[0] - b R θ[0] - b L s θ[0] - J R s θ[0] - J L s² θ[0] + R s ατ θ[0] + L s² ατ θ[0] - J R θ′[0] - J L s θ′[0] + R ατ θ′[0] + L s ατ θ′[0]) /
      (s (Ke Kt + b R + b L s + J R s + J L s² - R s ατ - L s² ατ))), LaplaceTransform[τ[t], t, s] →
  -((-b Kt L i[0] - J Kt L s i[0] + Kt L s ατ i[0] - b Kt LaplaceTransform[ea[t], t, s] - J Kt s LaplaceTransform[ea[t], t, s] +
       Kt s ατ LaplaceTransform[ea[t], t, s] + Ke Kt LaplaceTransform[τa[t], t, s] +
       J Ke Kt θ′[0] - Ke Kt ατ θ′[0]) / (Ke Kt + b R + b L s + J R s + J L s² - R s ατ - L s² ατ))}
```

```
{LaplaceTransform[e[t], t, s] → (Ke (Kt LaplaceTransform[ea[t], t, s] + (R + L s) LaplaceTransform[τa[t], t, s])) /
   (Ke Kt + (R + L s) (b + s (J - ατ))), LaplaceTransform[i[t], t, s] →
  ((b + s (J - ατ)) LaplaceTransform[ea[t], t, s] - Ke LaplaceTransform[τa[t], t, s]) / (Ke Kt + (R + L s) (b + s (J - ατ))),
 LaplaceTransform[θ[t], t, s] → (Kt LaplaceTransform[ea[t], t, s] + (R + L s) LaplaceTransform[τa[t], t, s]) /
   (s (Ke Kt + (R + L s) (b + s (J - ατ)))), LaplaceTransform[τ[t], t, s] →
  (Kt ((b + s (J - ατ)) LaplaceTransform[ea[t], t, s] - Ke LaplaceTransform[τa[t], t, s])) / (Ke Kt + (R + L s) (b + s (J - ατ)))}
```

We separate the transforms into applied-voltage- and applied-torque-dependent parts.

```
Clear[separate, decompose, inputLabels]
separate[xform_] := Module[{xformVarRules, invertedRules, xformVars, vard},
  xformVarRules = # → Unique["laplaceTransform"] & /@ inputXforms;
  invertedRules = Reverse /@ xformVarRules;
  xformVars = xformVarRules[[All, 2]];
  vard = xform /. xformVarRules;
  Collect[vard, xformVars] /. invertedRules
 ]
decompose[sum_] := {sum[[1]] / voltageXform, sum[[2]] / τinXform}
inputLabels = {"ea", "τa"}

solvedXforms = #[[1]] → separate[#[[2]]] & /@ rawSolvedXforms;
% // ColumnForm
```

```
{ea, τa}
```

$$\text{LaplaceTransform}[e[t], t, s] \to \frac{\text{Ke Kt LaplaceTransform}[\text{ea}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}+\frac{\text{Ke}\,(R+L\,s)\,\text{LaplaceTransform}[\tau\text{a}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}$$
$$\text{LaplaceTransform}[i[t], t, s] \to \frac{(b+s\,(J-\alpha\tau))\,\text{LaplaceTransform}[\text{ea}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}-\frac{\text{Ke LaplaceTransform}[\tau\text{a}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}$$
$$\text{LaplaceTransform}[\theta[t], t, s] \to \frac{\text{Kt LaplaceTransform}[\text{ea}[t],t,s]}{s\,(\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau)))}+\frac{(R+L\,s)\,\text{LaplaceTransform}[\tau\text{a}[t],t,s]}{s\,(\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau)))}$$
$$\text{LaplaceTransform}[\tau[t], t, s] \to \frac{\text{Kt}\,(b+s\,(J-\alpha\tau))\,\text{LaplaceTransform}[\text{ea}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}-\frac{\text{Ke Kt LaplaceTransform}[\tau\text{a}[t],t,s]}{\text{Ke Kt}+(R+L\,s)\,(b+s\,(J-\alpha\tau))}$$

Transfer Function Models

We use the solved transforms to create TransferFunctionModel[]s for every variable of interest.

```
Clear[makeModel]
makeModel[var_] := makeModel[var, ToString[var], 1]
makeModel[var_, outputLabel_, factor_] := TransferFunctionModel[
  factor * Function[{sum}, decompose[sum]]
    [LaplaceTransform[var[t], t, s] /. solvedXforms],
  s,
  SystemsModelLabels → {inputLabels, outputLabel}]
```

```
motorPositionModel = makeModel[θ]
motorVelocityModel = makeModel[θ, "Ω", s]
motorAccelerationModel = makeModel[θ, "α", s * s]
motorCurrentModel = makeModel[i]
motorEMFModel = makeModel[e]
motorTorqueModel = makeModel[τ]
```

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline \theta & \frac{Kt}{s\,(Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s))} & \frac{R+L\,s}{s\,(Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s))} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline \Omega & \frac{Kt}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} & \frac{R+L\,s}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline \alpha & \frac{Kt\,s}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} & \frac{s\,(R+L\,s)}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline i & \frac{b+(J-\alpha\tau)\,s}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} & -\frac{Ke}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline e & \frac{Ke\,Kt}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} & \frac{Ke\,(R+L\,s)}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline \tau & \frac{Kt\,(b+(J-\alpha\tau)\,s)}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} & -\frac{Ke\,Kt}{Ke\,Kt+(R+L\,s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

Time Domain Functions

The time domain functions we seek model various states of the motor as a direct function of time. Generally speaking, we obtain them from the frequency domain by (matrix) multiplying the transfer function by the Laplace transform of the input signal, then taking the inverse Laplace transform of the result. However, if we're starting with the (frequency domain) transfer function and the (time domain) input signal in-hand, we can compute the time domain function more efficiently using the Laplace transform convolution theorem.

Time Domain Output Functions

makeTimeDomainFunctionLaplace[]

makeTimeDomainFunctionLaplace[] computes the time domain function using the approach of obtaining everything in frequency domain then converting to time domain as a final step.

```
Clear[transformInputFunctions, makeTimeDomainFunctionLaplace]
transformInputFunctions[tInputFunctions_List, t_, s_] := LaplaceTransform[#[t], t, s] & /@ tInputFunctions
makeTimeDomainFunctionLaplace[model_, tInputFunctions_] := Module[
  {s = Unique["s"], t = Unique["t"], sInputs, sOutput, tOutput},
  sInputs = transformInputFunctions[tInputFunctions, t, s];
  sOutput = model[s] . sInputs;
  tOutput = InverseLaplaceTransform[sOutput, s, t];
  tOutput /. t → # &
 ]
```

makeTimeDomainFunctionConvolve[]

makeTimeDomainFunctionConvolve[] uses convolution and the theorem that the Laplace transform of a convolution is the product of the Laplace transform of the factors. Ref1: http://mathfaculty.fullerton.edu/mathews/c2003/laplaceconvolutionmod.html.

With the convolution approach, we can make explicit use of the fact that in the uses we have we know we want to take the inverse Laplace transform of a product, one of whose factors we already know the time domain inverse for, and thus we can avoid having LaplaceTransform[] do as much heavy lifting as we ask it to do in the other approach.

```
Clear[makeTimeDomainFunctionConvolve, convolve]
convolve[fExpr_, gExpr_, exprVar_, t_] := Module[{τ = Unique["τ"]},
  Assuming[t ≥ 0, Integrate[(fExpr /. exprVar → τ) * (gExpr /. exprVar → t - τ), {τ, 0, t}]]
 ]

makeTimeDomainFunctionConvolve[model_, tInputFunctions_] := Module[
  {s = Unique["s"], τ = Unique["τ"], modelExpr},
  modelExpr = InverseLaplaceTransform[model[s], s, τ][[1]];
  Function[{t}, Module[{convolutions},
    convolutions = convolve[modelExpr[[#]], (tInputFunctions[[#]])[τ], τ, t] &/@ Range[1, Length[tInputFunctions]];
    { Total[convolutions] } (* put in list to mirror InverseLaplaceTransform[] output structure *)
   ]
  ]
 ]
```

Small Tests

A few tests comparing both approaches.

```
testModel = TransferFunctionModel[
  {LaplaceTransform[1, t, s], LaplaceTransform[(Sin[t] + 7 Exp[-t]), t, s]}, s, SystemsModelLabels → {inputLabels, "out"}]

{(makeTimeDomainFunctionConvolve @@ #)[t], (makeTimeDomainFunctionLaplace @@ #)[t]} &/@ {
   {TransferFunctionModel[1 / (s^2 + 1), s], {2 Cos[#] &} (*Example 12.29 in Ref1*)},
   {testModel, {ν &, T &}}
  } // ColumnForm
```

$$\left(\begin{array}{c|cc} & ea & \tau a \\ \hline out & \frac{1}{s} & \frac{8+s+7 s^2}{(1+s)\,(1+s^2)} \end{array}\right)^{\mathcal{T}}$$

```
{{t Sin[t]}, {t Sin[t]}}
{{t ν + T (8 - 7 e^-t - Cos[t])}, {t ν + 8 T - 7 e^-t T - T Cos[t]}}
```

makeMotorTimeDomainFunction[]

makeMotorTimeDomainFunction[] takes a transfer function model and returns a function that, when provided with ea and τa time-domain input functions, returns a function of time.

```
Clear[makeMotorTimeDomainFunction]
makeMotorTimeDomainFunction[model_] := makeMotorTimeDomainFunction[model, makeTimeDomainFunctionConvolve]
makeMotorTimeDomainFunction[model_, builder_] := Module[{ea = Unique["ea"], τa = Unique["τa"], t = Unique["t"], expr},
  expr = builder[model, {(*1&, *)ea[#] &, τa[#] &}][t];
  Function[{eaActual, τaActual}, Module[{exprT},
    exprT = expr /. {ea → eaActual, τa → τaActual};
    Function[{tActual}, exprT /. t → tActual]]
  ]]
```

Larger Tests

We compare makeTimeDomainFunctionLaplace[] and makeTimeDomainFunctionConvolve[] on substantially-sized transfer functions. We see that, symbolically, for these tests they produce identical results, as they should. However, the makeTimeDomainFunctionConvolve[] approach is significantly faster, as therein we are asking InverseLaplaceTransform[] to significantly less heavy lifting. Moreover, and related to that point, the convolution approach is able to successfully compute time domain functions for some transfer functions and inputs that stymie makeTimeDomainFunctionLaplace[].

```
exampleConvolveE = makeMotorTimeDomainFunction[motorVelocityModel, makeTimeDomainFunctionConvolve][ν &, 0 &][t] // Timing
exampleConvolveτ = makeMotorTimeDomainFunction[motorVelocityModel, makeTimeDomainFunctionConvolve][0 &, 1 &][t] // Timing
```

```
{10.0156, {(Kt (1/(Ke Kt + b R)

   (√(b^2 L^2 + J^2 R^2 + 2 b L R ατ + ατ (4 Ke Kt L + R^2 ατ) - 2 J (2 Ke Kt L + R (b L + R ατ)))) + 2 e^(-(t (b L + J R - R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ))))/(2 L (J - ατ)))

    L (J - ατ) (1/(b L + J R - R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ))) +

      e^((t √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ)))/(L (J - ατ))) / (-b L - J R + R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ +

          R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ)))))) ν) / (√((b L + R (J - ατ))^2 - 4 L (Ke Kt + b R) (J - ατ)))}}
```

```
{14.0469, {(1/(Ke Kt + b R) 2 R (J - ατ) √(b^2 L^2 + J^2 R^2 + 2 b L R ατ + ατ (4 Ke Kt L + R^2 ατ) - 2 J (2 Ke Kt L + R (b L + R ατ))) +

   2 e^(-(t (b L + J R - R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ))))/(2 L (J - ατ))) L (J - ατ) ((e^((t √(b^2 L^2 + J^2 R^2 + 2 b L R ατ + ατ (4 Ke Kt L + R^2 ατ) - 2 J (2 Ke Kt L + R (b L + R ατ))))/(L (J - ατ)))

        (-b L + J R - R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ)))) /

      (-b L - J R + R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ))) -
     (b L - J R + R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ))) /

      (b L + J R - R ατ + √(b^2 L^2 + J^2 R^2 + 4 Ke Kt L ατ + 2 b L R ατ + R^2 ατ^2 - 2 J (2 Ke Kt L + b L R + R^2 ατ)))) /

  (2 √((b L + R (J - ατ))^2 - 4 L (Ke Kt + b R) (J - ατ)) (J - ατ))}}
```

```
exampleLaplaceE = makeMotorTimeDomainFunction[motorVelocityModel, makeTimeDomainFunctionLaplace][ν &, 0 &][t] // Timing
exampleLaplaceτ = makeMotorTimeDomainFunction[motorVelocityModel, makeTimeDomainFunctionLaplace][0 &, 1 &][t] // Timing
```

$$
\begin{aligned}
\Bigg\{39.3438, \Bigg\{\mathrm{Kt}\Bigg(\frac{1}{\mathrm{Ke}\,\mathrm{Kt}+b\,R} - \Bigg(
&-b\, e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} L + b\, e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} L - \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} J R + e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} J R + \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} R\,\alpha\tau - e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} R\,\alpha\tau + \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} \sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2} + \\
&e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} \sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}\Bigg) \Big/ \\
&\left(2(\mathrm{Ke}\,\mathrm{Kt}+b R)\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}\right)\Bigg)\nu\Bigg\}\Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
\Bigg\{39.6406, \\
\Bigg\{\frac{R}{\mathrm{Ke}\,\mathrm{Kt}+b\,R} - \Bigg(
&2\, e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} \mathrm{Ke}\,\mathrm{Kt}\, L - 2\, e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} \mathrm{Ke}\,\mathrm{Kt}\, L + \\
&b\, e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} L R - b\, e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} L R - \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} J R^2 + e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} J R^2 + \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} R^2\alpha\tau - e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} R^2\alpha\tau + \\
&e^{t\left(-\frac{b}{2(J-\alpha\tau)}-\frac{J R}{2 L (J-\alpha\tau)}+\frac{R\alpha\tau}{2 L (J-\alpha\tau)}-\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2 L (J-\alpha\tau)}\right)} R\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2} + \\
&e^{t\left(-\frac{b L}{2(J L-L\alpha\tau)}-\frac{J R}{2(J L-L\alpha\tau)}+\frac{R\alpha\tau}{2(J L-L\alpha\tau)}+\frac{\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)}\right)} R\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}\Bigg) \Big/ \\
&\left(2(\mathrm{Ke}\,\mathrm{Kt}+b R)\sqrt{-4(\mathrm{Ke}\,\mathrm{Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}\right)\Bigg\}\Bigg\}
\end{aligned}
$$

At first glance, the two approaches look like they might be producing quite different results. However, a little bit of work verifies that the applied-voltage functions are the same.

```
exampleConvolveE[[2, 1]] // Factor // Simplify
exampleLaplaceE[[2, 1]] // Factor // Simplify
%% / %
```

$$-\left(\left(e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,Kt\left(-b\,L+b\,e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,L-J\,R+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-2\,e^{\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+R\,\alpha\tau-e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,R\,\alpha\tau\right)\nu\right)\Big/\left(2\,(Ke\,Kt+b\,R)\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}\right)\right)$$

$$-\left(\left(e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,Kt\left(-b\,L+b\,e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,L-J\,R+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-2\,e^{\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+R\,\alpha\tau-e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\,R\,\alpha\tau\right)\nu\right)\Big/\left(2\,(Ke\,Kt+b\,R)\sqrt{(b\,L+R\,(J-\alpha\tau))^2-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}\right)\right)$$

```
1
```

That the applied-torque functions are equivalent is a little harder to verify, but we do (finally) manage it with a bit more work than we needed for the applied-voltage functions.

```
convolvedFull = exampleConvolveτ[[2, 1]] // Expand // FullSimplify // Expand // FullSimplify
laplaceFull  = exampleLaplaceτ[[2, 1]] // Expand // FullSimplify // Expand // FullSimplify
convolvedFull / laplaceFull
```

$$\frac{1}{2\,(Ke\,Kt+b\,R)}\left(2\,R-e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,R-e^{\frac{t\left(-b\,L-J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,R+\left(e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\left(-1+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\right)(2\,Ke\,Kt\,L+R\,(b\,L+R\,(-J+\alpha\tau)))\right)\Big/\left(\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}\right)\right)$$

$$\frac{1}{2\,(Ke\,Kt+b\,R)}\left(2\,R-e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,R-e^{\frac{t\left(-b\,L-J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}+R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\,R+\left(e^{-\frac{t\left(b\,L+J\,R+\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}-R\,\alpha\tau\right)}{2\,L\,(J-\alpha\tau)}}\left(-1+e^{\frac{t\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}}{L\,(J-\alpha\tau)}}\right)(2\,Ke\,Kt\,L+R\,(b\,L+R\,(-J+\alpha\tau)))\right)\Big/\left(\sqrt{(b\,L+R\,(J-\alpha\tau))^{2}-4\,L\,(Ke\,Kt+b\,R)\,(J-\alpha\tau)}\right)\right)$$

```
1
```

Huzzah. Going forward, we will use the convolution approach.

Time Domain Output Functions for Motor State

We define time domain functions for all the motor state of interest.

```
Clear[motorPosition, motorVelocity, motorAcceleration, motorCurrent, motorEMF, motorTorque]
```

```
motorPosition = makeMotorTimeDomainFunction[motorPositionModel];
```

```
motorVelocity = makeMotorTimeDomainFunction[motorVelocityModel];
```

```
motorAcceleration = makeMotorTimeDomainFunction[motorAccelerationModel];
```

```
motorCurrent = makeMotorTimeDomainFunction[motorCurrentModel];
```

```
motorEMF = makeMotorTimeDomainFunction[motorEMFModel];
```

```
motorTorque = makeMotorTimeDomainFunction[motorTorqueModel];
```

Time Domain Input Functions

Keeping track of units

In analogy to parameterQuantities above, we define inputQuantities, which adds units to our input parameters.

```
parameterQuantities
```

```
⟨|R → R Ω , L → L H , θ[t] → θ[t] rad , θ′[t] → θ′[t] rad/s , Ω[t] → Ω[t] rad/s , θ″[t] → θ″[t] rad/s^2 , α[t] → α[t] rad/s^2 ,
 i[t] → i[t] A , i′[t] → i′[t] A/s , i″[t] → i″[t] A/s^2 , ea[t] → ea[t] V , e[t] → e[t] V , τ[t] → τ[t] mN/rad ,
 τa[t] → τa[t] mN/rad , ατ → ατ m s^2N/rad^2 , J → J kg m^2/rad^2 , Ke → Ke s V/rad , Kt → Kt mN/(A rad) , b → b m s N/rad^2 |⟩
```

```
inputQuantities = {
   ea → Quantity[ea, "Volts"],
   τa → Quantity[τa, siTorqueUnits],
   cτ → Quantity[τa, siTorqueUnits],
   t → Quantity[t, "Seconds"]
  }
```

```
{ea → ea V , τa → τa mN/rad , cτ → τa mN/rad , t → t s}
```

Mass on pulley

Here, we model a mass suspended on a massless “rigid string” from a massless pulley attached to the motor shaft. That string acts at a wrench with a length of the radius of the pulley to deliver a torque to the shaft. τMassOnPulley[] computes that torque.

There is a force pulling down on the mass due to gravity and up due to acceleration in the string that results from the driven angular acceleration of the shaft. The sum of these forces is the tension in the string. If the net acceleration is negative (ie: downward), a real (flexible) string would clamp at zero and have the mass in free-fall; we choose not to model that for now. Instead, here, our hypothetical rigid string continues to impart acceleration from the pulley even for negative accelerations. This is a bit odd for a string, but more reasonable if the string were in fact a rigid arm being lifted, or some such.

```
Clear[τMassOnPulley]
τMassOnPulley[mass_, radius_][t_] := Module[
   {gravityAcceleration, gravityForce, angularAcceleration,
    tangentialAcceleration, tangentialForce, tension, torque, unit, parts, radians = Quantity["Radians"]},

   (* gravity exerts a constant force *)
   gravityAcceleration = Quantity["StandardAccelerationOfGravity"];
   gravityForce =  gravityAcceleration * mass;

   (* angular accelearation is resisted by mass, creates tangential force resisting acceleration *)
   angularAcceleration = θ''[t] /. parameterQuantities;
   tangentialAcceleration = angularAcceleration * radius / radians;
   tangentialForce = tangentialAcceleration * mass;

   tension = gravityForce + tangentialForce;

   (*tension acts like a wrench with lever arm 'radius'*)
   torque = tension * radius / radians;
   unit =  QuantityUnit[torque];
   parts = List @@ (QuantityMagnitude[torque] // Apart);
   {cτ → Quantity[parts[[1]], unit], ατ → Quantity[parts[[2]], unit] / angularAcceleration}
  ]
τMassOnPulley[Quantity[weightLbs, "Pounds"], Quantity[rInches, "Inches"]][t]
% // siUnits // clearUnits // N
```

```
{cτ → (196133 rInches weightLbs / 508) lb in^2/(s^2 rad) , ατ → rInches^2 weightLbs lb in^2/rad^2 }
```

```
{cτ → 0.112985 rInches weightLbs, ατ → 0.00029264 rInches^2 weightLbs}
```

Typical Input Examples

In analogy to aMotor above, we now define some typical inputs. The first has no externally applied torque.

```
anInput = Module[{},
   {
    t → Quantity[t, "Seconds"],
    ea → Quantity[12, "Volts"],
    τa → Quantity[0, siTorqueUnits],
    cτ → Quantity[0, siTorqueUnits],
    ατ → Quantity[0, (ατ /. parameterUnits)]
   }] // Association
```

```
⟨|t → t s , ea → 12 V , τa → 0 m N/rad , cτ → 0 m N/rad , ατ → 0 m s^2 N/rad^2 |⟩
```

A second has external torque from a mass on a pulley.

```
τMassOnPulley[Quantity[3, "Pounds"], Quantity[2, "Inches"]][t] // N
τMassOnPulley[Quantity[3, "Pounds"], Quantity[2, "Inches"]][t] // siUnits // N
anInputPulley = {t → Quantity[t, "Seconds"], ea → Quantity[12, "Volts"]} ~Join~
   Module[{pulley = τMassOnPulley[Quantity[3, "Pounds"], Quantity[2, "Inches"]]},
    Print[pulley[t]];
    pulley = pulley[t] // siUnits;
    {cτ → (cτ /. pulley), τa → (cτ /. pulley), ατ → (ατ /. pulley)}] // Association
```

```
{cτ → 2316.53 lb in^2/(s^2 rad) , ατ → 12. lb in^2/rad^2 }
```

```
{cτ → 0.677909 kg m^2/(s^2 rad) , ατ → 0.00351168 kg m^2/rad^2 }
```

```
{cτ → 588399/254 lb in^2/(s^2 rad) , ατ → 12 lb in^2/rad^2 }
```

```
⟨|t → t s , ea → 12 V , cτ → 3389544870828501/5000000000000000 kg m^2/(s^2 rad) ,
 τa → 3389544870828501/5000000000000000 kg m^2/(s^2 rad) , ατ → 2194797400719/625000000000000 kg m^2/rad^2 |⟩
```

A third leaves quantities with symbolic parameters

```
genericInputτ = Module[{},
   {
    t → Quantity[t, "Seconds"],
    ea → Quantity[ea, "Volts"],
    τa → Quantity[cτ, siTorqueUnits],
    cτ → Quantity[cτ, siTorqueUnits],
    ατ → Quantity[ατ, (ατ /. parameterUnits)]
   }] // Association
```

```
⟨|t → t s , ea → ea V , τa → cτ m N/rad , cτ → cτ m N/rad , ατ → ατ m s^2 N/rad^2 |⟩
```

Examples

We try out our time-domain functions on our example data. Notice how the units come out correctly: they flow automatically from beginning to end

```
anInput // N
anInputPulley // N
```

```
⟨|t → t s , ea → 12. V , τa → 0. m N/rad , cτ → 0. m N/rad , ατ → 0. m s^2 N/rad^2 |⟩
```

```
⟨|t → t s , ea → 12. V , cτ → 0.677909 kg m^2/(s^2 rad) , τa → 0.677909 kg m^2/(s^2 rad) , ατ → 0.00351168 kg m^2/rad^2 |⟩
```

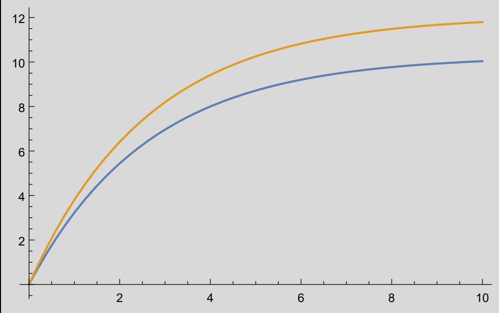
```
velUnits = motorVelocity[ea &, τa &][t] /. aMotor /. anInput // N // siUnits // FullSimplify
```

$\{e^{-0.377374\,t}\,(-10.2734\ \mathrm{rad/s}) + e^{-4754.7\,t}\,(0.000815385\ \mathrm{rad/s}) + 10.2726\ \mathrm{rad/s}\}$

```
vel = velUnits // clearUnits
velτ = motorVelocity[ea &, τa &][t] /. aMotor /. anInputPulley // N // siUnits // clearUnits // FullSimplify
Plot[{vel, velτ}, {t, 0, 10}]
```

$\{10.2726 + 0.000815385\, e^{-4754.7\,t} - 10.2734\, e^{-0.377374\,t}\}$

$\{12.0691 + 0.00081827\, e^{-4754.7\,t} - 12.0699\, e^{-0.378704\,t}\}$



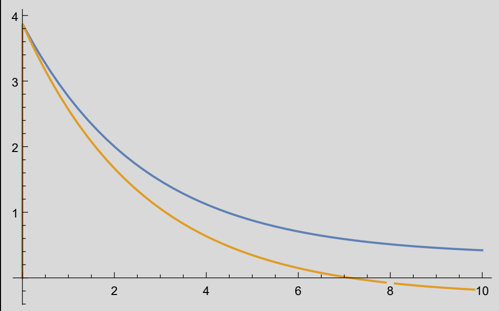
Good: the model with the (positive) external torque achieves greater velocity, as it should.

```
torUnits = motorTorque[ea &, τa &][t] /. aMotor /. anInput // N // siUnits // FullSimplify
torτ = motorTorque[ea &, τa &][t] /. aMotor /. anInputPulley // N // siUnits // clearUnits // FullSimplify
tor = torUnits // clearUnits
Plot[{tor, torτ}, {t, 0, 10}]
```

$\{e^{-4754.7\,t}\,(-3.87693\ \mathrm{kg\,m^2/(s^2 rad)}) + 0.338995\ \mathrm{kg\,m^2/(s^2 rad)} + e^{-0.377374\,t}\,(3.53793\ \mathrm{kg\,m^2/(s^2 rad)})\}$

$\{-0.279629 - 3.87698\, e^{-4754.7\,t} + 4.15661\, e^{-0.378704\,t}\}$

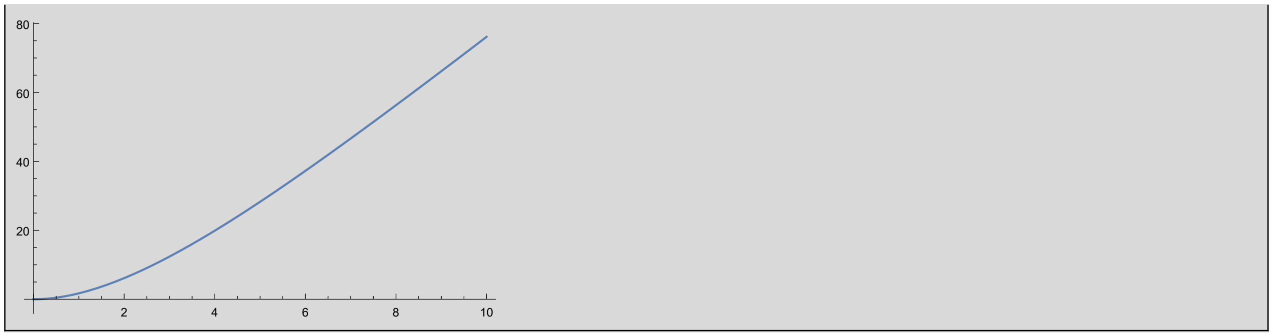
$\{0.338995 - 3.87693\, e^{-4754.7\,t} + 3.53793\, e^{-0.377374\,t}\}$



```
posUnits = motorPosition[ea &, τa &][t]  /. aMotor /. anInput // N // siUnits // FullSimplify
pos = posUnits // clearUnits
Plot[{pos}, {t, 0, 10}]
```

$\{e^{-4754.7\,t}\,(-1.7149\times10^{-7}\ \mathrm{rad}) + e^{-0.377374\,t}\,(27.2234\ \mathrm{rad}) + (-27.2234 + 10.2726\,t)\ \mathrm{rad}\}$

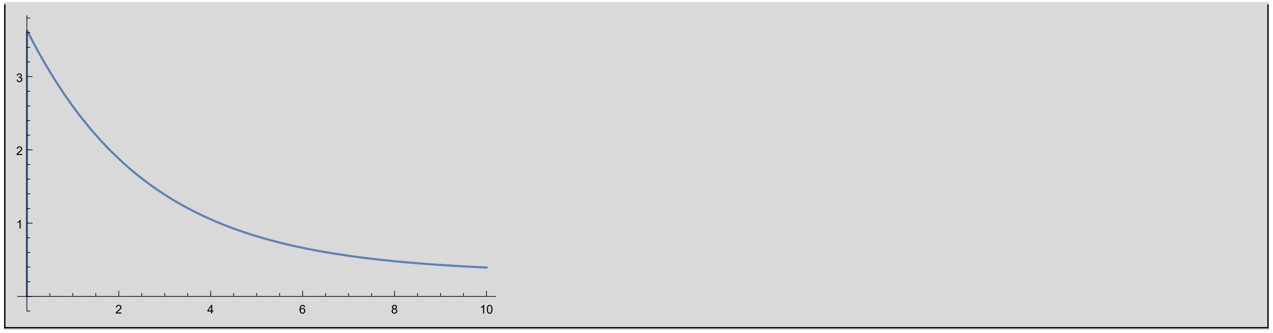
$\{-27.2234 - 1.7149\times10^{-7}\,e^{-4754.7\,t} + 27.2234\,e^{-0.377374\,t} + 10.2726\,t\}$



```
curUnits = motorCurrent[ea &, τa &][t]  /. aMotor /. anInput // N // siUnits // FullSimplify
cur = curUnits // clearUnits
Plot[{cur}, {t, 0, 10}]
```

$\{e^{-4754.7\,t}\,(-3.63689\ \mathrm{A}) + 0.318007\ \mathrm{A} + e^{-0.377374\,t}\,(3.31888\ \mathrm{A})\}$

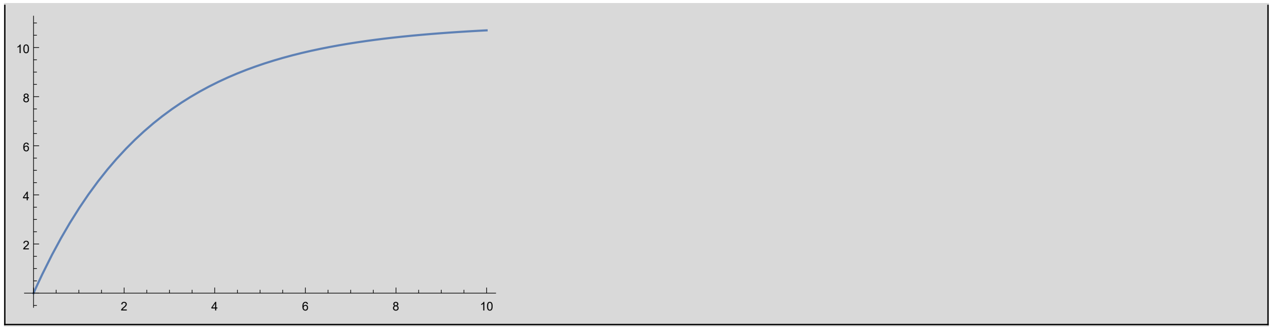
$\{0.318007 - 3.63689\,e^{-4754.7\,t} + 3.31888\,e^{-0.377374\,t}\}$



```
emfUnits = motorEMF[ea &, τa &][t]  /. aMotor /. anInput // N // siUnits // FullSimplify
emf = emfUnits // clearUnits
Plot[{emf}, {t, 0, 10}]
```

$\{e^{-0.377374\,t}\,(-10.9514\ \mathrm{kg\,m^2/(s^3 A)}) + e^{-4754.7\,t}\,(0.000869201\ \mathrm{kg\,m^2/(s^3 A)}) + 10.9506\ \mathrm{kg\,m^2/(s^3 A)}\}$

$\{10.9506 + 0.000869201\,e^{-4754.7\,t} - 10.9514\,e^{-0.377374\,t}\}$



Steady State

Calculating Steady State Values

Brute Force vs The Final Value Theorem

We explore the steady state behavior, the limit of the time domain functions as time goes to infinity.

One way to calculate said limit is to brute-force calculate the limit in question. Let’s explore that with velocity.

```
parameterAssumptions = (# ≥ 0 & /@ Complement[Keys[aMotor ~Join~ anInput], {τa, ατ}]) ~Join~ {τa ∈ ℝ, ατ ∈ ℝ}
genericVelExpr =
  motorVelocity[ea &, τa &][t] /. genericMotor /. genericInputτ // siUnits // clearUnits // clearUnits // FullSimplify
genericVelLimit = Limit[genericVelExpr, t → Infinity, Assumptions → parameterAssumptions]
```

```
{b ≥ 0, cτ ≥ 0, ea ≥ 0, J ≥ 0, Ke ≥ 0, Kt ≥ 0, L ≥ 0, R ≥ 0, t ≥ 0, τa ∈ ℝ, ατ ∈ ℝ}
```

$$\left\{\frac{1}{2\,(Ke\,Kt+b\,R)}\left(2\,(ea\,Kt+c\tau\,R)+2\,e^{-\frac{1}{2}t\left(\frac{R}{L}+\frac{b}{J-\alpha\tau}\right)}\left(-(ea\,Kt+c\tau\,R)\,\mathrm{Cosh}\left[\frac{1}{2\,J\,L-2\,L\,\alpha\tau}t\sqrt{\left(-4\,J\,Ke\,Kt\,L+b^2L^2-2\,b\,J\,L\,R+J^2R^2+4\,Ke\,Kt\,L\,\alpha\tau+2\,b\,L\,R\,\alpha\tau-2\,J\,R^2\,\alpha\tau+R^2\,\alpha\tau^2\right)}\right]+\left(\left(2\,c\tau\,Ke\,Kt\,L+b\,L\,(-ea\,Kt+c\tau\,R)+ea\,Kt\,R\,(-J+\alpha\tau)+c\tau\,R^2\,(-J+\alpha\tau)\right)\mathrm{Sinh}\left[\frac{1}{2\,J\,L-2\,L\,\alpha\tau}t\sqrt{\left(-4\,J\,Ke\,Kt\,L+b^2L^2-2\,b\,J\,L\,R+J^2R^2+4\,Ke\,Kt\,L\,\alpha\tau+2\,b\,L\,R\,\alpha\tau-2\,J\,R^2\,\alpha\tau+R^2\,\alpha\tau^2\right)}\right]\right)\Big/\left(\sqrt{\left(b^2L^2+J^2R^2+2\,b\,L\,R\,\alpha\tau+\alpha\tau\left(4\,Ke\,Kt\,L+R^2\,\alpha\tau\right)-2\,J\,(2\,Ke\,Kt\,L+R\,(b\,L+R\,\alpha\tau))\right)}\right)\right)\right)\right\}$$

$$\left\{\mathrm{ConditionalExpression}\left[\frac{ea\,Kt+c\tau\,R}{Ke\,Kt+b\,R},\ \left(\sqrt{\left(-4\,J\,Ke\,Kt\,L+b^2L^2-2\,b\,J\,L\,R+J^2R^2+4\,Ke\,Kt\,L\,\alpha\tau+2\,b\,L\,R\,\alpha\tau-2\,J\,R^2\,\alpha\tau+R^2\,\alpha\tau^2\right)}\ \middle|\ 1\Big/\left(\sqrt{\left(b^2L^2+J^2R^2+2\,b\,L\,R\,\alpha\tau+\alpha\tau\left(4\,Ke\,Kt\,L+R^2\,\alpha\tau\right)-2\,J\left(2\,Ke\,Kt\,L+b\,L\,R+R^2\,\alpha\tau\right)\right)}\right)\right)\in\mathbb{R}\ \&\&\ L\,(J-\alpha\tau)\,(b\,L+J\,R-R\,\alpha\tau)>0\ \&\&\ L\,(J-\alpha\tau)\sqrt{\left(-4\,J\,Ke\,Kt\,L+b^2L^2-2\,b\,J\,L\,R+J^2R^2+4\,Ke\,Kt\,L\,\alpha\tau+2\,b\,L\,R\,\alpha\tau-2\,J\,R^2\,\alpha\tau+R^2\,\alpha\tau^2\right)}>0\ \&\&\ L\,(J-\alpha\tau)\left(b\,L+J\,R-R\,\alpha\tau-\sqrt{\left(-4\,J\,Ke\,Kt\,L+b^2L^2-2\,b\,J\,L\,R+J^2R^2+4\,Ke\,Kt\,L\,\alpha\tau+2\,b\,L\,R\,\alpha\tau-2\,J\,R^2\,\alpha\tau+R^2\,\alpha\tau^2\right)}\right)>0\right]\right\}$$

A more efficient way to calculate the steady state behavior is to avail ourselves of the use of the Final Value Theorem (Ref: https://en.wikipedia.org/wiki/-Final_value_theorem). The final value theorem tells us that if the time-domain limit exists, it is equal to the limit of the Laplace transform of the time domain function as s → 0.

```
Clear[ssFinalValueTheorem]
ssFinalValueTheorem[model_, inputs_] := Module[{s = Unique[], t = Unique[], expr},
   expr = model[s].(LaplaceTransform[#[t], t, s] & /@ inputs);
   Limit[s expr, s → 0][[1]]
 ]
```

```
(ss = {
    ssPos → ssFinalValueTheorem[motorPositionModel, {ea &, cτ &}] // Factor,
    ssVel → ssFinalValueTheorem[motorVelocityModel, {ea &, cτ &}] // Factor,
    ssAcc → ssFinalValueTheorem[motorAccelerationModel, {ea &, cτ &}] // Factor,
    ssEmf → ssFinalValueTheorem[motorEMFModel, {ea &, cτ &}] // Factor,
    ssCur → ssFinalValueTheorem[motorCurrentModel, {ea &, cτ &}] // Factor,
    ssTor → ssFinalValueTheorem[motorTorqueModel, {ea &, cτ &}] // Factor
   }) // prettyPrint
```

$$\begin{aligned}
&\text{ssPos} \to \text{Indeterminate}\\
&\text{ssVel} \to \frac{\text{ea Kt}+c\tau R}{\text{Ke Kt}+b R}\\
&\text{ssAcc} \to 0\\
&\text{ssEmf} \to \frac{\text{Ke}\,(\text{ea Kt}+c\tau R)}{\text{Ke Kt}+b R}\\
&\text{ssCur} \to \frac{b\,\text{ea}-c\tau\,\text{Ke}}{\text{Ke Kt}+b R}\\
&\text{ssTor} \to \frac{(b\,\text{ea}-c\tau\,\text{Ke})\,\text{Kt}}{\text{Ke Kt}+b R}
\end{aligned}$$

We observe that, for velocity, at least, the Final Value Theorem approach gives the same value as does the brute force approach. Huzzah! Moreover, computing the Final Value Theorem values is much more efficient.

On the Applicability of the Final Value Theorem

Can we always use the Final Value Theorem?

No, the Final Value Theorem only applies under certain conditions. Specifically: that all non-zero poles of the transfer function must have negative real parts, and the transfer function must have at most one pole at the origin (Ref: http://www-personal.umich.edu/~dsbaero/others/39-FVTrevisited.pdf). The predicate exhibited in the ConditionalExpression of the brute-force result captures exactly these conditions. When using the Final Value Theorem on any concrete system, we will need to test the conditions in any particular application. Lets illustrate by looking at the velocity model.

```
motorVelocityModel
(velPoles = TransferFunctionPoles[motorVelocityModel]) // MatrixForm
```

$$\left(\begin{array}{c|cc} & \text{ea} & \tau\text{a} \\ \hline \Omega & \dfrac{\text{Kt}}{\text{Ke Kt}+(R+L s)\,(b+(J-\alpha\tau)\,s)} & \dfrac{R+L s}{\text{Ke Kt}+(R+L s)\,(b+(J-\alpha\tau)\,s)} \end{array}\right)^{\mathcal{T}}$$

$$\left(\begin{array}{cc} \left(\begin{array}{c} \frac{-b L-J R+R\alpha\tau-\sqrt{-4(\text{Ke Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)} \\ \frac{-b L-J R+R\alpha\tau+\sqrt{-4(\text{Ke Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)} \end{array}\right) & \left(\begin{array}{c} \frac{-b L-J R+R\alpha\tau-\sqrt{-4(\text{Ke Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)} \\ \frac{-b L-J R+R\alpha\tau+\sqrt{-4(\text{Ke Kt}+b R)(J L-L\alpha\tau)+(b L+J R-R\alpha\tau)^2}}{2(J L-L\alpha\tau)} \end{array}\right) \end{array}\right)$$

The poles are the same for both applied-voltage and applied-torque input since the transfer function denominator is the same for both inputs, in both cases being the zeros of:

```
den = motorVelocityModel[s][[1, 2]] // Denominator
```

$$\text{Ke Kt}+(R+L s)\,(b+s\,(J-\alpha\tau))$$

In order to get a feel for where these zeros are in practice, let’s look at that denominator for an actual example motor. We compute where the poles occur as a function of $\alpha\tau$.

```
polesατ = den /. aMotor /. parameterQuantities // siUnits // clearUnits
Solve[polesατ == 0, s] // N
```

$$\frac{284089}{250000}+\left(\frac{33}{10}+\frac{347\,s}{500000}\right)\left(\frac{33}{1000}+s\left(\frac{100001041}{100000000}-\alpha\tau\right)\right)$$

$$\begin{aligned}
&\Big\{\Big\{s \to \Big(14.4092\,\Big(1.65003\times10^{10}-1.65\times10^{10}\,\alpha\tau-1.\,\sqrt{2.72173\times10^{20}-5.44423\times10^{20}\,\alpha\tau+2.7225\times10^{20}\,\alpha\tau^2}\Big)\Big)\Big/\left(-1.00001\times10^{8}+1.\times10^{8}\,\alpha\tau\right)\Big\},\\
&\ \Big\{s \to \Big(14.4092\,\Big(1.65003\times10^{10}-1.65\times10^{10}\,\alpha\tau+\sqrt{2.72173\times10^{20}-5.44423\times10^{20}\,\alpha\tau+2.7225\times10^{20}\,\alpha\tau^2}\Big)\Big)\Big/\left(-1.00001\times10^{8}+1.\times10^{8}\,\alpha\tau\right)\Big\}\Big\}
\end{aligned}$$

By exploring the system a bit, we can see that for certain values of $\alpha\tau$ (in this one example, apparently any $\alpha\tau$ bigger than one), poles exist in the right hand plane, and thus the Final Value Theorem does not apply:

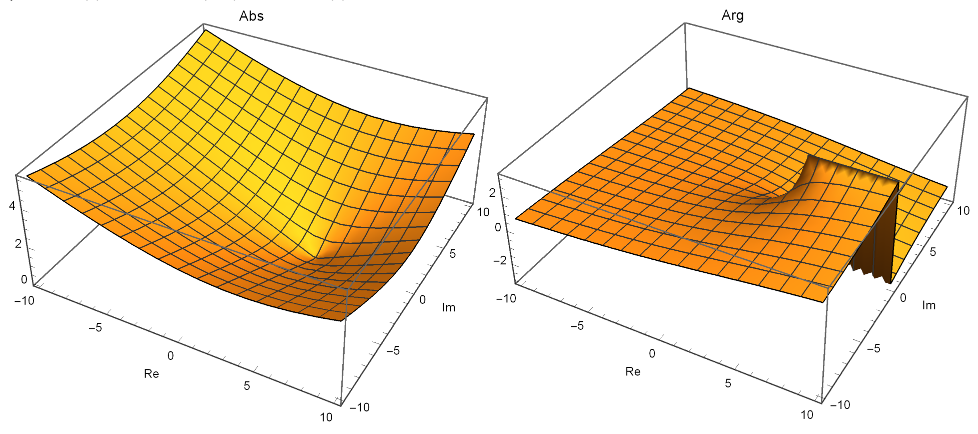
```
Manipulate[
 Column[{polesατ /. {ατ → var},
   poles → (s /. Solve[Evaluate[polesατ /. {ατ → var}] == 0, s] // ReIm),
   Row[
    {Plot3D[polesατ /. {s → x + I y, ατ → var} // Abs, {x, -range, range},
      {y, -range, range}, ImageSize → Medium, AxesLabel → {"Re", "Im"}, PlotLabel → "Abs"],
     Plot3D[polesατ /. {s → x + I y, ατ → var} // Arg, {x, -range, range}, {y, -range, range},
      ImageSize → Medium, AxesLabel → {"Re", "Im"}, PlotLabel → "Arg"] }
   ]} ],
 {{var, 1.1, "ατ"}, -10, 10}, {{range, 10}, 0, 10000}]
```

ατ

range

$\frac{284089}{250000} + \left(\frac{33}{1000} - 0.0999896\, s\right)\left(\frac{33}{10} + \frac{347\, s}{500000}\right)$

```
poles → {{-4758.48, 0}, {3.77117, 0}}
```



Using the “hard-way” limit previously calculated, we can verify the observation the steady state velocity is unbounded for such $\alpha\tau$.

```
genericVelLimit /. aMotor /. parameterQuantities // siUnits // clearUnits // N
% /. ατ → # & /@ {0, 1/2, 1, 2}
```

```
{ConditionalExpression[0.803048 (3.3 cτ + 1.066 ea),
  (√(10.8869 - 21.7769 ατ + 10.89 ατ^2) | 1/(√(10.8902 + 0.000151153 ατ - 2.00002 (0.00165284 + 10.89 ατ) + ατ (0.00315452 + 10.89 ατ)))) ∈
   ℝ && 0.000694 (3.30006 - 3.3 ατ) (1.00001 - 1. ατ) > 0. && 0.000694 (1.00001 - 1. ατ) √(10.8869 - 21.7769 ατ + 10.89 ατ^2) > 0. &&
  0.000694 (1.00001 - 1. ατ) (3.30006 - 3.3 ατ - 1. √(10.8869 - 21.7769 ατ + 10.89 ατ^2)) > 0.]}
```

```
{{0.803048 (3.3 cτ + 1.066 ea)}, {0.803048 (3.3 cτ + 1.066 ea)}, {Undefined}, {Undefined}}
```

An analogous analytic development awaits a future revision to this document.

```
Collect[den // Expand, s]
```

```
Ke Kt + b R + s^2 (J L - L ατ) + s (b L + J R - R ατ)
```

Back EMF vs Applied Voltage

We want to know the steady-state velocity at which the back EMF balances the input voltage. Thus, we need EMF in terms of speed. We have EMF from voltage, and speed from voltage. So we need to invert the latter, then compose.

```
Clear[ssAppliedVoltageFromVelocity]
ssAppliedVoltageFromVelocity[velocity_] := Module[{eqn, velSym = Unique["vel"]},
  eqn = velSym == (ssVel /. ss);
  ea /. uniqueSolve[eqn, ea][[1]] /. velSym → velocity
 ]
ssAppliedVoltageFromVelocity[Ω]
```

$$\frac{-c\tau R + Ke\,Kt\,\Omega + b\,R\,\Omega}{Kt}$$

```
Clear[ssEmfFromAppliedVoltage, ssEmfFromVelocity]
ssEmfFromAppliedVoltage[voltage_] := (ssEmf /. ss) /. ea → voltage
ssEmfFromVelocity[velocity_] := ssEmfFromAppliedVoltage[ssAppliedVoltageFromVelocity[velocity]]
ssEmfFromVelocity[Ω]
% // FullSimplify
```

$$\frac{Ke\,(Ke\,Kt\,\Omega + b\,R\,\Omega)}{Ke\,Kt + b\,R}$$

$$Ke\,\Omega$$

Well, that’s a result now, isn’t it? Of course, this is expected, given the underlying differential equations:

```
diffEqns[[1]]
```

$$e[t] == Ke\,\theta'[t]$$

However, that we arrive at the same result the long-way around boosts confidence that the intervening math is in fact correct.

So, at what velocity does the applied voltage balance the back EMF?

```
ssEmfFromVelocity[Ω] == ssAppliedVoltageFromVelocity[Ω] // FullSimplify
uniqueSolve[%, Ω]
emfThresholdVelocity = %[[1]]
```

$$\frac{R\,(c\tau - b\,\Omega)}{Kt} == 0$$

$$\left\{\Omega \to \frac{c\tau}{b}\right\}$$

$$\Omega \to \frac{c\tau}{b}$$

That’s interesting: the equalizing velocity is dependent on the motor (through its viscous drag parameter), the external system (through the externally applied torque), but *not* the externally applied voltage. Also: if there is no external torque, then the threshold velocity is zero, which is as it should be, since in that situation the back EMF can never match the externally applied voltage due to losses except when all is stopped.

A quick check that the units are correct:

```
emfThresholdVelocity /. parameterQuantities /. inputQuantities
```

$$\Omega \to \frac{\tau a}{b}\,\text{rad/s}$$

What does this look like for an actual motor with our example weight attached?

```
(Ω /. emfThresholdVelocity) /. aMotor /. anInputPulley // siUnits
UnitConvert[%, "rpm"] // N
UnitConvert[%, "Revolutions / Second"] * Quantity[1120, IndependentUnit["ticks"] / "Revolutions"]
```

$$\frac{102\,713\,480\,934\,197}{5\,000\,000\,000\,000}\ \mathrm{rad/s}$$

```
196.168 rev/min
```

```
3661.81 ticks/s
```

Revision History

- 2018.07.23 Added acceleration-dependant external torque. Corrected “viscous friction” to “viscous drag”. Clarified usage of units.