magnetforces

1. About this file. This is a 'literate programming' approach to writing Matlab code using MATLABWEB[1]. To be honest I don't know if it's any better than simply using the Matlab programming language directly. The big advantage for me is that you have access to the entire LaTeX document environment, which gives you access to vastly better tools for cross-referencing, maths typesetting, structured formatting, bibliography generation, and so on.

The downside is obviously that you miss out on Matlab's IDE with its integrated M-Lint program, debugger, profiler, and so on. Depending on ones work habits, this may be more or less of limiting factor to using 'literate programming' in this way.

2. Calculating forces between magnets. This is the source to some code to calculate the forces (and perhaps torques) between two cuboid-shaped magnets with arbitary displacement and magnetisation direction.

If this code works then I'll look at calculating the forces for magnets with rotation as well.

⟨`magnetforces.m` 2⟩ ≡

function [*forces_out*] =*magnetforces*(*magnet_fixed*, *magnet_float*, *magnet_disp*)
  ⟨Matlab help text 21⟩
  ⟨Extract input variables 3⟩
  ⟨Decompose orthogonal superpositions 4⟩
  ⟨Calculate all forces 5⟩
  end
  ⟨Functions for calculating forces and stiffnesses 15⟩

[1] http://tug.ctan.org/pkg/matlabweb

3. First of all, address the data structures required for the input and output. Because displacement of a single magnet has three components, plus sizes of the faces another three, plus magnetisation strength and direction (two) makes nine in total, we use one of Matlab's structures to pass the information into the function. Otherwise we'd have an overwhelming number of input arguments.

We use spherical coordinates to represent magnetisation angle, where *phi* is the angle from the horizontal plane ($-\pi/2 \le \phi \le \pi/2$) and θ is the angle around the horizontal plane ($0 \le \theta \le 2\pi$). This follows Matlab's definition; other conventions are commonly used as well. Remember:

$$
\begin{aligned}
&(1,0,0)_{\text{cartesian}} \equiv (0,0,1)_{\text{spherical}} \\
&(0,1,0)_{\text{cartesian}} \equiv (\pi/2,0,1)_{\text{spherical}} \\
&(0,0,1)_{\text{cartesian}} \equiv (0,\pi/2,1)_{\text{spherical}}
\end{aligned}
$$

⟨Extract input variables 3⟩ ≡

$a_1 = 0.5 * magnet\_fixed.dim(1);$
$b_1 = 0.5 * magnet\_fixed.dim(2);$
$c_1 = 0.5 * magnet\_fixed.dim(3);$
$a_2 = 0.5 * magnet\_float.dim(1);$
$b_2 = 0.5 * magnet\_float.dim(2);$
$c_2 = 0.5 * magnet\_float.dim(3);$

$J1r = magnet\_fixed.magn;$
$J2r = magnet\_float.magn;$
$J1t = magnet\_fixed.magdir(1);$
$J2t = magnet\_float.magdir(1);$
$J1p = magnet\_fixed.magdir(2);$
$J2p = magnet\_float.magdir(2);$

$dx = magnet\_disp(1);$
$dy = magnet\_disp(2);$
$dz = magnet\_disp(3);$

This code is used in section 2.

4. Superposition is used to turn an arbitrary magnetisation angle into a set of orthogonal magnetisations.

Each magnet can potentially have three components, which can result in up to nine force calculations for a single magnet.

We don't use Matlab's sph2cart here, because it doesn't calculate zero accurately (because it uses radians).

$\langle$ Decompose orthogonal superpositions 4 $\rangle \equiv$

$$
\begin{aligned}
&J1x = J1r*cosd(J1p)*cosd(J1t);\\
&J2x = J2r*cosd(J2p)*cosd(J2t);\\
&J1y = J1r*cosd(J1p)*sind(J1t);\\
&J2y = J2r*cosd(J2p)*sind(J2t);\\
&J1z = J1r*sind(J1p);\\
&J2z = J2r*sind(J2p);\\
&J1 = [J1x\ \ J1y\ \ J1z];\\
&J2 = [J2x\ \ J2y\ \ J2z];
\end{aligned}
$$

This code is used in section 2.

5. The expressions we have to calculate the forces assume a fixed magnet with positive z magnetisation only. Secondly, magnetisation direction of the floating magnet may only be in the positive z- or y-directions.

The parallel forces are more easily visualised; if *J1z* is negative, then transform the coordinate system so that up is down and down is up. Then proceed as usual and reverse the vertical forces in the last step.

The orthogonal forces require reflection and/or rotation to get the displacements in a form suitable for calculation.

Initialise a $3 \times 3 \times 3$ array to store each force component in each direction, and fill it up by calculating

⟨ Calculate all forces 5 ⟩ ≡

```
force_components = repmat(NaN, [3 3 3]);
⟨ Precompute rotation matrices 20 ⟩
disp('x-x');
⟨ Calculate forces x-x 9 ⟩
disp('x-y');
⟨ Calculate forces x-y 12 ⟩
disp('x-z');
⟨ Calculate forces x-z 14 ⟩
disp('y-x');
⟨ Calculate forces y-x 13 ⟩
disp('y-y');
⟨ Calculate forces y-y 10 ⟩
disp('y-z');
⟨ Calculate forces y-z 11 ⟩
disp('z-x');
⟨ Calculate forces z-x 8 ⟩
disp('z-y');
⟨ Calculate forces z-y 7 ⟩
disp('z-z');
⟨ Calculate forces z-z 6 ⟩
forces_out = squeeze(sum(sum(force_components, 1), 2));
```

This code is used in section 2.

6. The easy one. Note that *J1z* and *J2z* can be negative and the forces calculated will be correct.

⟨ Calculate forces z-z 6 ⟩ ≡

```
[Fx Fy Fz] = forces_parallel(a1, b1, c1, a2, b2, c2, [dx, dy, dz], J1, J2);
force_components(3, 3, :) = [Fx Fy Fz];
```

This code is used in section 5.

7. Now the analogous calculation with the orthogonal force. I'm hoping that the correct forces are calculated even if the magnetisations are negative.

$\langle$ Calculate forces z-y 7 $\rangle \equiv$

$[Fx\ Fy\ Fz] = \mathit{forces\_orthogonal}(a_1, b_1, c_1, a_2, b_2, c_2, [dx, dy, dz], J1, J2);$
$\mathit{force\_components}(3, 2, :) = [Fx\ Fy\ Fz];$

This code is used in section 5.

8. For x magnetisation, we can just calculate it *as is* it were in the y direction, ensuring to swap the necessary components.

$\langle$ Calculate forces z-x 8 $\rangle \equiv$

$[Fy\ Fx\ Fz] = \mathit{forces\_orthogonal}(b_1, a_1, c_1, b_2, a_2, c_2, [dy, dx, dz], J1, J2(1));$
$\mathit{force\_components}(3, 1, :) = [Fx\ Fy\ Fz];$

This code is used in section 5.

9. The other parallel forces (x-x and y-y) require a rotation to get the magnetisations correctly aligned. In this case, rotate the entire coordinate system around the y-axis so that the new z is the old x. After the forces are calculated, rotate them back to the original coordinate system.

$\langle$ Calculate forces x-x 9 $\rangle \equiv$

$drot = \mathit{rotate\_x\_to\_z}([dx\ dy\ dz]);$
$J1rot = \mathit{rotate\_x\_to\_z}(J1);$
$J2rot = \mathit{rotate\_x\_to\_z}(J2);$
$[Fx\ Fy\ Fz] = \mathit{forces\_parallel}(c_1, b_1, a_1, c_2, b_2, a_2, drot, J1rot, J2rot);$
$\mathit{force\_components}(1, 1, :) = \mathit{rotate\_z\_to\_x}([Fx\ Fy\ Fz]);$

This code is used in section 5.

10. Same again, this time making y the 'up' direction.

$\langle$ Calculate forces y-y 10 $\rangle \equiv$

$drot = \mathit{rotate\_y\_to\_z}([dx\ dy\ dz]);$
$J1rot = \mathit{rotate\_y\_to\_z}(J1);$
$J2rot = \mathit{rotate\_y\_to\_z}(J2);$
$[Fx\ Fy\ Fz] = \mathit{forces\_parallel}(a_1, c_1, b_1, a_2, c_2, b_2, drot, J1rot, J2rot);$
$\mathit{force\_components}(2, 2, :) = \mathit{rotate\_z\_to\_y}([Fx\ Fy\ Fz]);$

This code is used in section 5.

11. There are four more force calculations. y-z is z-y rotated.

⟨ Calculate forces y-z 11 ⟩ ≡

$drot = rotate\_y\_to\_z([dx\ \ dy\ \ dz]);$
$J1rot = rotate\_y\_to\_z(J1);$
$J2rot = rotate\_y\_to\_z(J2);$
$[Fx\ \ Fy\ \ Fz] = forces\_orthogonal(a_1, c_1, b_1, a_2, c_2, b_2, drot, J1rot, J2rot);$
$force\_components(2, 3, :) = rotate\_z\_to\_y([Fx\ \ Fy\ \ Fz]);$

This code is used in section 5.

12. x-y is z-y rotated.

⟨ Calculate forces x-y 12 ⟩ ≡

$drot = rotate\_x\_to\_z([dx\ \ dy\ \ dz]);$
$J1rot = rotate\_x\_to\_z(J1);$
$J2rot = rotate\_x\_to\_z(J2);$
$[Fx\ \ Fy\ \ Fz] = forces\_orthogonal(c_1, b_1, a_1, c_2, b_2, a_2, drot, J1rot, J2rot);$
$force\_components(1, 2, :) = rotate\_z\_to\_x([Fx\ \ Fy\ \ Fz]);$

This code is used in section 5.

13. The last two are more difficult. y-x is z-x rotated; z-x is z-y with swapped x and y components.

⟨ Calculate forces y-x 13 ⟩ ≡

$drot = swap\_x\_y(rotate\_y\_to\_z([dx\ \ dy\ \ dz]));$
$J1rot = swap\_x\_y(rotate\_y\_to\_z(J1));$
$J2rot = swap\_x\_y(rotate\_y\_to\_z(J2));$
$[Fx\ \ Fy\ \ Fz] = forces\_orthogonal(c_1, a_1, b_1, c_2, a_2, b_2, drot, J1rot, J2rot);$
$force\_components(2, 1, :) = rotate\_z\_to\_y(swap\_x\_y([Fx\ \ Fy\ \ Fz]));$

This code is used in section 5.

14. x-z is z-x rotated; z-x is z-y with swapped x and y components.

⟨ Calculate forces x-z 14 ⟩ ≡

$drot = swap\_x\_y(rotate\_x\_to\_z([dx\ \ dy\ \ dz]));$
$J1rot = swap\_x\_y(rotate\_x\_to\_z(J1));$
$J2rot = swap\_x\_y(rotate\_x\_to\_z(J2));$
$[Fx\ \ Fy\ \ Fz] = forces\_orthogonal(b_1, c_1, a_1, b_2, c_2, a_2, drot, J1rot, J2rot);$
$force\_components(1, 3, :) = rotate\_z\_to\_x(swap\_x\_y([Fx\ \ Fy\ \ Fz]));$

This code is used in section 5.

15. Functions for calculating forces and stiffnesses. The calculations for forces between differently-oriented cuboid magnets are all directly from the literature. The stiffnesses have been derived by differentiating the force expressions, but that's the easy part.

⟨ Functions for calculating forces and stiffnesses 15 ⟩ ≡

⟨ Parallel magnets force calculation 16 ⟩
⟨ Parallel magnets stiffness calculation 17 ⟩
⟨ Orthogonal magnets force calculation 18 ⟩
⟨ Orthogonal magnets stiffness calculation 19 ⟩

This code is used in section 2.

16. The expressions here follow directly from Akoun and Yonnet [1].

Inputs:	(a, b, c)	the half dimensions of the fixed magnet
	(A, B, C)	the half dimensions of the floating magnet
	(dx, dy, dz)	distance between magnet centres
	$(J, J2)$	magnetisations of the magnet(s) in the z-direction
Outputs:	(Fx, Fy, Fz)	Forces of the second magnet

⟨Parallel magnets force calculation 16⟩ ≡

```
function [Fx Fy Fz] =forces_parallel(a, b, c, A, B, C, offset, J1, J2)
        % You probably want to call
        % warning off MATLAB:divideByZero
        % warning off MATLAB:log:logOfZero
  if length(J1) ≡ 3
    J1 = J1(3);
  end
  if length(J2) ≡ 3
    J2 = J2(3);
  end
  if  ( J1 ≡ 0 OR J2 ≡ 0 )
  disp('Zero magnetisation (parallel)')
  Fx = 0;
  Fy = 0;
  Fz = 0;
  return;
  end
  dx = offset(1);
  dy = offset(2);
  dz = offset(3);
  [index_h, index_j, index_k, index_l, index_p, index_q] = ndgrid([0 1]);
  index_sum = (−1) .^ (index_h + index_j + index_k + index_l + index_p +
       index_q);
        % (Using this method is actually LESS efficient than using six for
        % loops for h..q over [0 1], but it looks a bit nicer, huh?)
  u = dx + A∗(−1) .^ index_j − a∗(−1) .^ index_h;
  v = dy + B∗(−1) .^ index_l − b∗(−1) .^ index_k;
  w = dz + C∗(−1) .^ index_q − c∗(−1) .^ index_p;
  r = sqrt(u .^ 2 + v .^ 2 + w .^ 2);
  f_x = ...
  +0.5∗(v .^ 2 − w .^ 2) .∗ log(r − u) ...
  +u .∗ v .∗ log(r − v) ...
  +v .∗ w .∗ atan(u .∗ v ./ r ./ w) ...
  +0.5∗r .∗ u;
```

$f\_y = \ldots$
$+0.5*(u\ .\hat{}\ 2 - w\ .\hat{}\ 2)\ .*\log(r - v)\ldots$
$+u\ .*\ v\ .*\log(r - u)\ldots$
$+u\ .*\ w\ .*\ \mathrm{atan}(u\ .*\ v\ ./\ r\ ./\ w)\ldots$
$+0.5*r\ .*\ v;$

$f\_z = \ldots$
$-u\ .*\ w\ .*\log(r - u)\ldots$
$-v\ .*\ w\ .*\log(r - v)\ldots$
$+u\ .*\ v\ .*\ \mathrm{atan}(u\ .*\ v\ ./\ r\ ./\ w)\ldots$
$-r\ .*\ w;$

$fx = index\_sum\ .*\ f\_x;$
$fy = index\_sum\ .*\ f\_y;$
$fz = index\_sum\ .*\ f\_z;$

$magconst = J1*J2/(4*\pi*(4*\pi*1\cdot 10^{-7}));$
$Fx = magconst*\mathrm{sum}(fx(:));$
$Fy = magconst*\mathrm{sum}(fy(:));$
$Fz = magconst*\mathrm{sum}(fz(:));$

end

This code is used in section 15.

17. And these are the stiffnesses.

Inputs:	(a, b, c)	the half dimensions of the fixed magnet
	(A, B, C)	the half dimensions of the floating magnet
	(dx, dy, dz)	distance between magnet centres
	$(J, J2)$	magnetisations of the magnet(s) in the z-direction
Outputs:	(Kx, Ky, Kz)	Stiffnesses of the 2nd magnet

⟨Parallel magnets stiffness calculation 17⟩ ≡

```
function [Kx Ky Kz] = stiffness_parallel(a, b, c, A, B, C, dx, dy, dz, J, J2)
    % You probably want to call
    % warning off MATLAB:divideByZero
    % warning off MATLAB:log:logOfZero
  if ( J ≡ 0 OR J2 ≡ 0 )
  Kx = 0;
  Ky = 0;
  Kz = 0;
  return;
  end
  [index_h, index_j, index_k, index_l, index_p, index_q] = ndgrid([0 1]);
  index_sum = (−1) .^ (index_h + index_j + index_k + index_l + index_p +
      index_q);
    % Using this method is actually less efficient than using six for
    % loops for h..q over [0 1]. To be addressed.
  u = dx + A∗(−1) .^ index_j − a∗(−1) .^ index_h;
  v = dy + B∗(−1) .^ index_l − b∗(−1) .^ index_k;
  w = dz + C∗(−1) .^ index_q − c∗(−1) .^ index_p;
  r = sqrt(u .^ 2 + v .^ 2 + w .^ 2);
  k_x = ...
  −r ...
  −(u .^ 2.∗v) ./ (u .^ 2 + w .^ 2) ...
  −v .∗ log(r − v);
  k_y = ...
  −r ...
  −(v .^ 2.∗u) ./ (v .^ 2 + w .^ 2) ...
  −u .∗ log(r − u);
  k_z = −k_x − k_y;
  kx = index_sum .∗ k_x;
  ky = index_sum .∗ k_y;
  kz = index_sum .∗ k_z;
  magconst = J∗J2/(4∗π∗(4∗π∗1 · 10^−7));
  Kx = magconst∗sum(kx(:));
  Ky = magconst∗sum(ky(:));
  Kz = magconst∗sum(kz(:));
```

end

This code is used in section 15.

18. Orthogonal magnets forces given by Yonnet and Allag [2]. The magnetisation of the floating magnet *J2* is in the positive y-direction.

⟨ Orthogonal magnets force calculation 18 ⟩ ≡

```
function [Fx Fy Fz] = forces_orthogonal(a, b, c, A, B, C, offset, J1, J2)
  if length(J1) ≡ 3
    J1 = J1(3);
  end
  if length(J2) ≡ 3
    J2 = J2(2);
  end
  if ( J1 ≡ 0 OR J2 ≡ 0 )
  disp('Zero␣magnetisation␣(orth)')
  Fx = 0;
  Fy = 0;
  Fz = 0;
  return;
  end
  dx = offset(1);
  dy = offset(2);
  dz = offset(3);
  [index_h, index_j, index_k, index_l, index_p, index_q] = ndgrid([0 1]);
  index_sum = (−1) .^ (index_h + index_j + index_k + index_l + index_p +
      index_q);
      % (Using this method is actually LESS efficient than using six for
      % loops for h..q over [0 1], but it looks a bit nicer, huh?)
  u = dx + A∗(−1) .^ index_j − a∗(−1) .^ index_h;
  v = dy + B∗(−1) .^ index_l − b∗(−1) .^ index_k;
  w = dz + C∗(−1) .^ index_q − c∗(−1) .^ index_p;
  r = sqrt(u .^ 2 + v .^ 2 + w .^ 2);
  f_x = ...
  −v .∗ w .∗ log(r − u) ...
  +v .∗ u .∗ log(r + w) ...
  +w .∗ u .∗ log(r + v) ...
  −0.5∗u .^ 2 .∗ atan(v .∗ w ./ (u .∗ r)) ...
  −0.5∗v .^ 2 .∗ atan(u .∗ w ./ (v .∗ r)) ...
  −0.5∗w .^ 2 .∗ atan(u .∗ v ./ (w .∗ r));
  f_y = ...
  0.5∗(u .^ 2 − v .^ 2) .∗ log(r + w) ...
  −u .∗ w .∗ log(r − u) ...
  −u .∗ v .∗ atan(u .∗ w ./ (v .∗ r)) ...
  −0.5∗w .∗ r;
  f_z = ...
  0.5∗(u .^ 2 − w .^ 2) .∗ log(r + v) ...
```

$-u\ .{*}\ v\ .{*}\ \log(r-u)\ldots$
$-u\ .{*}\ w\ .{*}\ \mathrm{atan}(u\ .{*}\ v\ ./\ (w\ .{*}\ r))\ldots$
$-0.5{*}v\ .{*}\ r;$

$fx = index\_sum\ .{*}\ f\_x;$
$fy = index\_sum\ .{*}\ f\_y;$
$fz = index\_sum\ .{*}\ f\_z;$

$magconst = J1{*}J2/(4{*}\pi{*}(4{*}\pi{*}1\cdot 10^{-7}));$
$Fx = magconst{*}\mathrm{sum}(fx(:));$
$Fy = magconst{*}\mathrm{sum}(fy(:));$
$Fz = magconst{*}\mathrm{sum}(fz(:));$

end

This code is used in section 15.

19. Orthogonal magnets stiffnesses.

$\langle$ Orthogonal magnets stiffness calculation 19 $\rangle \equiv$

```
% not yet calculated
```

This code is used in section 15.

20. When the forces are rotated we use these rotation matrices to avoid having to think too hard. Use degrees in order to compute $sin\pi/2$ exactly!

$\langle$ Precompute rotation matrices 20 $\rangle \equiv$

$Rx = @(\theta)\ [1\ 0\ 0;\ \ 0\ cosd(\theta) - sind(\theta);\ \ 0\ sind(\theta)\ cosd(\theta)];$
$Ry = @(\theta)\ [cosd(\theta)\ 0\ sind(\theta);\ \ 0\ 1\ 0;\ \ -sind(\theta)\ 0\ cosd(\theta)];$
$Rz = @(\theta)\ [cosd(\theta) - sind(\theta)\ 0;\ \ sind(\theta)\ cosd(\theta)\ 0;\ \ 0\ 0\ 1];$

$rotate\_z\_to\_x = @(vec)\ Ry(90){*}vec';$
$rotate\_x\_to\_z = @(vec)\ Ry(-90){*}vec';$

$rotate\_z\_to\_y = @(vec)\ Rx(-90){*}vec';$
$rotate\_y\_to\_z = @(vec)\ Rx(90){*}vec';$

$swap\_x\_y = @(vec)\ [vec(2)\ vec(1)\ vec(3)];$

This code is used in section 5.

21. When users type `help magnetforces` this is what they see. This is designed to be displayed in a fixed-width font so the output here will be fairly ugly.

$\langle$ Matlab help text 21 $\rangle \equiv$

```
%% MAGNETFORCES  Calculate forces between two cuboid magnets
%
% Finish this off later.
%
```

This code is used in section 2.

22. Test files. The chunks that follow are designed to be saved into individual files and executed automatically to check for (a) correctness and (b) regression problems as the code evolves.

How do I know if the code produces the correct forces? Well, for many cases I can compare with published values in the literature. Beyond that, I'll be setting up some tests that I can logically infer should produce the same results (such as mirror-image displacements) and test that.

There are many Matlab unit test frameworks but I'll be using a fairly low-tech method. In time this test suite should be (somehow) useable for all implementations of `magnetocode`, not just Matlab.

⟨`magforce_test001a.m` 22⟩ ≡

```
f = [];
magnet_fixed.dim = [0.04 0.04 0.04];
magnet_float.dim = magnet_fixed.dim;
magnet_fixed.magn = 1.3;
magnet_float.magn = magnet_fixed.magn;
magnet_fixed.magdir = [0 90];      % vertical
magnet_float.magdir = magnet_fixed.magdir;
displ = [0 0 0.1];
magnet_float.magn = 1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
magnet_float.magn = -1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
magnet_fixed.magdir = [0 0];      % x
magnet_float.magdir = magnet_fixed.magdir;
displ = [0.1 0 0];
magnet_float.magn = 1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
magnet_float.magn = -1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
magnet_fixed.magdir = [90 0];      % y
magnet_float.magdir = magnet_fixed.magdir;
displ = [0 0.1 0];
magnet_float.magn = 1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
magnet_float.magn = -1.3;
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, displ + eps);
f( : , end +1 ) = magnetforces(magnet_fixed, magnet_float, -displ + eps);
```

$assert(chop(f(3, 1), 6) \equiv -chop(f(3, 2), 6));$
$assert(chop(f(3, 2), 6) \equiv chop(f(3, 3), 6));$
$assert(chop(f(3, 3), 6) \equiv -chop(f(3, 4), 6));$

disp('Tests␣passed');

23. Get some numbers:

⟨`magforce_test002a.m` 23⟩ ≡

$magnet\_fixed.dim = [0.02\ 0.04\ 0.06];$
$magnet\_fixed.magn = 1.3;$
$magnet\_fixed.magdir = [90\ 0] + eps;$ % vertical

$magnet\_float.dim = [0.07\ 0.05\ 0.03];$
$magnet\_float.magn = 1.1;$
$magnet\_float.magdir = [90\ 0] + eps;$ % vertical

$magnet\_disp = [0.1\ 0.15\ 0.05];$

Index of `magnetforces`

List of Refinements in `magnetforces`

References

[1] Gilles Akoun and Jean-Paul Yonnet. "3D analytical calculation of the forces exerted between two cuboidal magnets". In: *IEEE Transactions on Magnetics* MAG-20.5 (Sept. 1984), pp. 1962–1964. DOI: 10.1109/TMAG.1984.1063554.

[2] Jean-Paul Yonnet and Hicham Allag. "Analytical Calculation of Cubodal Magnet Interactions in 3D". In: *The 7th International Symposium on Linear Drives for Industry Application*. 2009.