GMLinear Core 1.0 API Reference

Format

Vectors

- An n-dimensional vector is represented as a 1D array of size n.
- In string form: Each entry is written with 14 decimal digits, separated by commas.
- In Base64 form: Each entry is written as a 64-bit float to the buffer.

Matrices

- An m by n matrix is represented as a 2D array of height m and length n.
- In string form: Each entry is written with 14 decimal digits, separated by commas. Rows are separated by semicolons.
- In Base64 form: Each entry is written as a 64-bit float to the buffer, in row-major order.

Constants

Index Constants

These constants allow you to semantically access vector entries.

Name	Value
COMP_X	**0**
COMP_Y	**1**
COMP_Z	**2**
COMP_T	**2**
COMP_W	**3**
COMP_RED	**0**
COMP_GREEN	**1**
COMP_BLUE	**2**
COMP_ALPHA	**3**

Constructors

`r2(x0, x1)`

Return a new 2D vector.

`r2_zeros()`

Return a new 2D zero vector.

`r3(x0, x1, x2)`

Return a new 3D vector.

`r3_zeros()`
Return a new 3D zero vector.

`r4(x0, x1, x2, x3)`
Return a new 4D vector.

`r4_zeros()`
Return a new 4D zero vector.

`rn(…)`
Return a new n-dimensional vector.

`rn_zeros(n)`
Return a new n-dimensional zero vector.

`r22(x00, x01, x10, x11)`
Return a new 2×2 matrix.

`r22_identity()`
Return a new 2×2 identity matrix.

`r22_zeros()`
Return a new 2×2 zero matrix.

`r33(x00, x01, x02, x10, x11, x12, x20, x21, x22)`
Return a new 3×3 matrix.

`r33_identity()`
Return a new 3×3 identity matrix.

`r33_zeros()`
Return a new 3×3 zero matrix.

`r44(x00, x01, x02, x03, x10, x11, x12, x13, x20, x21, x22, x23, x30, x31, x32, x33)`
Return a new 4×4 matrix.

`r44_identity()`
Return a new 4×4 identity matrix.

`r44_zeros()`
Return a new 4×4 zero matrix.

`rmn(m, n, …)`
Return a new m×n matrix with entries in row-major order.

`rmn_zeros(m, n)`
Return a new m×n zero matrix.

`rnn(…)`
Return a new n×n matrix with entries in row-major order. It will create a 1×1 matrix with 1 argument, a 2×2 matrix with 4 arguments, a 3×3 matrix with 9 arguments, or a 4×4 matrix with 16 arguments.

`rnn_identity()`
Return a new n×n identity matrix.

`rnn_zeros()`
Return a new n×n zero matrix.

2D Vector Operations

`r2_clone(v)`
Return a clone of 2D vector $\vec{v}$.

`r2_clone_to(v, vout)`
Copy $\vec{v}$ to $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_add(v1, v2)`
Return $\vec{v}_1 + \vec{v}_2$.

`r2_add_to(v1, v2, vout)`
Save the result of $\vec{v}_1 + \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_subtract(v1, v2)`
Return $\vec{v}_1 - \vec{v}_2$.

`r2_subtract_to(v1, v2, vout)`
Save the result of $\vec{v}_1 - \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_dot(v1, v2)`
Return the dot product $\vec{v}_1 \cdot \vec{v}_2$.

`r2_scale(v, r)`
Return the scalar product $r\vec{v}$.

`r2_scale_to(v, r, vout)`
Save the scalar product $r\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_unit(v)`
Return the unit vector in the direction of $\vec{v}$ (i.e. $\frac{1}{\|\vec{v}\|}\vec{v}$).

`r2_unit_to(v, vout)`
Save the unit vector in the direction of $\vec{v}$ into $\vec{v}_{out}$.

`r2_norm(v)`
Return $\|\vec{v}\|$ (Euclidean norm).

`r2_1norm(v)`
Return $\|\vec{v}\|_1$ (Manhattan norm).

`r2_maxnorm(v)`
Return $\|\vec{v}\|_\infty$ (Maximum norm).

`r2_lerp(v1, v2, amount)`
Return the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`r2_lerp_to(v1, v2, amount, vout)`
Save the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`r2_dist(v1, v2)`
Return the Euclidean distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|$).

`r2_1dist(v1, v2)`
Return the Manhattan distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|_1$).

`r2_proj(v1, v2)`
Return the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{proj}_{\vec{b}}\vec{a} = \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`r2_proj_to(v1, v2, vout)`
Save the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_rej(v1, v2)`
Return the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{rej}_{\vec{b}}\vec{a} = \vec{a} - \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`r2_rej_to(v1, v2, vout)`
Save the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_encode_string(v)`
Return the string form of $\vec{v}$.

`r2_decode_string(str)`
Return the vector represented by **`str`**.

`r2_decode_string_to(str, vout)`
Save the vector represented by **`str`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r2_encode_base64(v)`
Return the Base64 string form of $\vec{v}$.

`r2_decode_base64(enc)`
Return the vector represented by Base64 string **`enc`**.

`r2_decode_base64_to(enc, vout)`
Save the vector represented by Base64 string **`enc`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

3D Vector Operations

`r3_clone(v)`
Return a clone of 3D vector $\vec{v}$.

`r3_clone_to(v, vout)`
Copy $\vec{v}$ to $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_add(v1, v2)`
Return $\vec{v}_1 + \vec{v}_2$.

`r3_add_to(v1, v2, vout)`
Save the result of $\vec{v}_1 + \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_subtract(v1, v2)`
Return $\vec{v}_1 - \vec{v}_2$.

`r3_subtract_to(v1, v2, vout)`
Save the result of $\vec{v}_1 - \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_dot(v1, v2)`
Return the dot product $\vec{v}_1 \cdot \vec{v}_2$.

`r3_cross(v1, v2)`
Return the cross product $\vec{v}_1 \times \vec{v}_2$.

`r3_cross_to(v1, v2, vout)`
Save the cross product $\vec{v}_1 \times \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_scale(v, r)`
Return the scalar product $r\vec{v}$.

r3_scale_to(v, r, vout)

Save the scalar product $r\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

r3_unit(v)

Return the unit vector in the direction of $\vec{v}$ (i.e. $\frac{1}{\|\vec{v}\|}\vec{v}$).

r3_unit_to(v, vout)

Save the unit vector in the direction of $\vec{v}$ into $\vec{v}_{out}$.

r3_norm(v)

Return $\|\vec{v}\|$ (Euclidean norm).

r3_1norm(v)

Return $\|\vec{v}\|_1$ (Manhattan norm).

r3_maxnorm(v)

Return $\|\vec{v}\|_\infty$ (Maximum norm).

r3_lerp(v1, v2, amount)

Return the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

r3_lerp_to(v1, v2, amount, vout)

Save the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

r3_dist(v1, v2)

Return the Euclidean distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|$).

r3_1dist(v1, v2)

Return the Manhattan distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|_1$).

r3_proj(v1, v2)

Return the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{proj}_{\vec{b}}\vec{a} = \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

r3_proj_to(v1, v2, vout)

Save the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

r3_rej(v1, v2)

Return the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{rej}_{\vec{b}}\vec{a} = \vec{a} - \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

r3_rej_to(v1, v2, vout)

Save the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_encode_string(v)`
Return the string form of $\vec{v}$.

`r3_decode_string(str)`
Return the vector represented by **`str`**.

`r3_decode_string_to(str, vout)`
Save the vector represented by **`str`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r3_encode_base64(v)`
Return the Base64 string form of $\vec{v}$.

`r3_decode_base64(enc)`
Return the vector represented by Base64 string **`enc`**.

`r3_decode_base64_to(enc, vout)`
Save the vector represented by Base64 string **`enc`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

4D Vector Operations

`r4_clone(v)`
Return a clone of 4D vector $\vec{v}$.

`r4_clone_to(v, vout)`
Copy $\vec{v}$ to $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_add(v1, v2)`
Return $\vec{v}_1 + \vec{v}_2$.

`r4_add_to(v1, v2, vout)`
Save the result of $\vec{v}_1 + \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_subtract(v1, v2)`
Return $\vec{v}_1 - \vec{v}_2$.

`r4_subtract_to(v1, v2, vout)`
Save the result of $\vec{v}_1 - \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_dot(v1, v2)`
Return the dot product $\vec{v}_1 \cdot \vec{v}_2$.

`r4_scale(v, r)`
Return the scalar product $r\vec{v}$.

`r4_scale_to(v, r, vout)`

Save the scalar product $r\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_unit(v)`

Return the unit vector in the direction of $\vec{v}$ (i.e. $\frac{1}{\|\vec{v}\|}\vec{v}$).

`r4_unit_to(v, vout)`

Save the unit vector in the direction of $\vec{v}$ into $\vec{v}_{out}$.

`r4_norm(v)`

Return $\|\vec{v}\|$ (Euclidean norm).

`r4_1norm(v)`

Return $\|\vec{v}\|_1$ (Manhattan norm).

`r4_maxnorm(v)`

Return $\|\vec{v}\|_\infty$ (Maximum norm).

`r4_lerp(v1, v2, amount)`

Return the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`r4_lerp_to(v1, v2, amount, vout)`

Save the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`r4_dist(v1, v2)`

Return the Euclidean distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|$).

`r4_1dist(v1, v2)`

Return the Manhattan distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|_1$).

`r4_proj(v1, v2)`

Return the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{proj}_{\vec{b}}\vec{a} = \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`r4_proj_to(v1, v2, vout)`

Save the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_rej(v1, v2)`

Return the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{rej}_{\vec{b}}\vec{a} = \vec{a} - \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`r4_rej_to(v1, v2, vout)`

Save the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_encode_string(v)`
Return the string form of $\vec{v}$.

`r4_decode_string(str)`
Return the vector represented by **`str`**.

`r4_decode_string_to(str, vout)`
Save the vector represented by **`str`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r4_encode_base64(v)`
Return the Base64 string form of $\vec{v}$.

`r4_decode_base64(enc)`
Return the vector represented by Base64 string **`enc`**.

`r4_decode_base64_to(enc, vout)`
Save the vector represented by Base64 string **`enc`** into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

General Vector Operations

`rn_clone(v)`
Return a clone of n-dimensional vector $\vec{v}$.

`rn_clone_to(v, vout)`
Copy $\vec{v}$ to $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_add(v1, v2)`
Return $\vec{v}_1 + \vec{v}_2$.

`rn_add_to(v1, v2, vout)`
Save the result of $\vec{v}_1 + \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_subtract(v1, v2)`
Return $\vec{v}_1 - \vec{v}_2$.

`rn_subtract_to(v1, v2, vout)`
Save the result of $\vec{v}_1 - \vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_dot(v1, v2)`
Return the dot product $\vec{v}_1 \cdot \vec{v}_2$.

`rn_scale(v, r)`
Return the scalar product $r\vec{v}$.

`rn_scale_to(v, r, vout)`

Save the scalar product $r\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_unit(v)`

Return the unit vector in the direction of $\vec{v}$ (i.e. $\frac{1}{\|\vec{v}\|}\vec{v}$).

`rn_unit_to(v, vout)`

Save the unit vector in the direction of $\vec{v}$ into $\vec{v}_{out}$.

`rn_norm(v)`

Return $\|\vec{v}\|$ (Euclidean norm).

`rn_1norm(v)`

Return $\|\vec{v}\|_1$ (Manhattan norm).

`rn_maxnorm(v)`

Return $\|\vec{v}\|_\infty$ (Maximum norm).

`rn_lerp(v1, v2, amount)`

Return the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`rn_lerp_to(v1, v2, amount, vout)`

Save the linear interpolation between $\vec{v}_1$ and $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$. An amount of 0 corresponds to $\vec{v}_1$ and an amount of 1 corresponds to $\vec{v}_2$.

`rn_dist(v1, v2)`

Return the Euclidean distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|$).

`rn_1dist(v1, v2)`

Return the Manhattan distance between $\vec{v}_1$ and $\vec{v}_2$ (i.e. $\|\vec{v}_2 - \vec{v}_1\|_1$).

`rn_proj(v1, v2)`

Return the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{proj}_{\vec{b}}\vec{a} = \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`rn_proj_to(v1, v2, vout)`

Save the vector projection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_rej(v1, v2)`

Return the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ ($\text{rej}_{\vec{b}}\vec{a} = \vec{a} - \frac{\vec{a}\cdot\vec{b}}{\vec{b}\cdot\vec{b}}\vec{b}$)

`rn_rej_to(v1, v2, vout)`

Save the vector rejection of $\vec{v}_1$ onto $\vec{v}_2$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rn_encode_string(v)`
Return the string form of $\vec{v}$.

`rn_decode_string(str)`
Return the vector represented by **`str`**.

`rn_decode_string_to(str, vout)`
Save the vector represented by **`str`** into v_{out} and return $\vec{v}_{out}$.

`rn_encode_base64(v)`
Return the Base64 string form of $\vec{v}$.

`rn_decode_base64(enc, n)`
Return the vector represented by Base64 string **`enc`**.

`rn_decode_base64_to(enc, n, vout)`
Save the vector represented by Base64 string **`enc`** into v_{out} and return $\vec{v}_{out}$.

Coordinate Conversions
These functions are named in the form **`rN_IN_OUT(v)`** and **`rN_IN_OUT_to(v, vout)`**. See the tables below for a list of supported coordinate systems.

2D Coordinate Systems ($N = 2$)

Name	Name Code (IN/OUT)	Component 0	Component 1
Rectangular	**`rec`**	x (pixels)	y (pixels)
GameMaker Polar	**`gmp`**	len (pixels)	dir (degrees, counter-clockwise from right with downward y axis)
Polar	**`pol`**	r (pixels)	θ (radians, counter-clockwise from right with upward y axis)

3D Coordinate Systems ($N = 3$)

Name	Name Code (IN/OUT)	Component 0	Component 1	Component 2
Rectangular	**`rec`**	x (pixels)	y (pixels)	z (pixels)
Cylindrical	**`cyl`**	ρ (pixels)	ϕ (radians)	z (pixels)
Spherical	**`sph`**	ρ (pixels)	ϕ (radians, azimuth)	θ (radians, declination)

`r2_rec_gmp(v_xy)`
Return the GameMaker polar coordinate equivalent of rectangular coordinate $\vec{v}_{xy}$.

`r2_rec_gmp_to(vi_xy, vo_ld)`

Save the GameMaker polar coordinate equivalent of rectangular coordinate input $\vec{v}_{xy}$ into output $\vec{v}_{ld}$ and return $\vec{v}_{id}$.

`r2_gmp_rec(v_ld)`

Return the rectangular coordinate equivalent of GameMaker polar coordinate $\vec{v}_{ld}$.

`r2_gmp_rec_to(vi_ld, vo_xy)`

Save the rectangular coordinate equivalent of GameMaker polar coordinate input $\vec{v}_{ld}$ into output $\vec{v}_{xy}$ and return $\vec{v}_{xy}$.

`r2_rec_pol(v_xy)`

Return the polar coordinate equivalent of rectangular coordinate $\vec{v}_{xy}$.

`r2_rec_pol_to(vi_xy, vo_rt)`

Save the polar coordinate equivalent of rectangular coordinate input $\vec{v}_{xy}$ into output $\vec{v}_{r\theta}$ and return $\vec{v}_{r\theta}$.

`r2_pol_rec(v_rt)`

Return the rectangular coordinate equivalent of polar coordinate $\vec{v}_{r\theta}$.

`r2_pol_rec_to(vi_rt, vo_xy)`

Save the rectangular coordinate equivalent of polar coordinate input $\vec{v}_{r\theta}$ into output $\vec{v}_{xy}$ and return $\vec{v}_{xy}$.

`r2_gmp_pol(v_ld)`

Return the polar coordinate equivalent of GameMaker polar coordinate $\vec{v}_{ld}$.

`r2_gmp_pol_to(vi_ld, vo_rt)`

Save the polar coordinate equivalent of GameMaker polar coordinate input $\vec{v}_{ld}$ into output $\vec{v}_{r\theta}$ and return $\vec{v}_{r\theta}$.

`r2_pol_gmp(v_rt)`

Return the GameMaker polar coordinate equivalent of polar coordinate $\vec{v}_{r\theta}$.

`r2_pol_gmp_to(vi_rt, vo_ld)`

Save the GameMaker polar coordinate equivalent of polar coordinate input $\vec{v}_{r\theta}$ into output $\vec{v}_{ld}$ and return $\vec{v}_{id}$.

`r3_rec_cyl(v_xyz)`

Return the cylindrical coordinate equivalent of rectangular coordinate $\vec{v}_{xyz}$.

`r3_rec_cyl_to(vi_xyz, vo_rpz)`

Save the cylindrical coordinate equivalent of rectangular coordinate input $\vec{v}_{xyz}$ into output $\vec{v}_{r\phi z}$ and return $\vec{v}_{r\phi z}$.

`r3_cyl_rec(v_rpz)`
Return the rectangular coordinate equivalent of cylindrical coordinate $\vec{v}_{r\phi z}$.

`r3_cyl_rec_to(vi_rpz, vo_xyz)`
Save the rectangular coordinate equivalent of cylindrical coordinate input $\vec{v}_{r\phi z}$ into output $\vec{v}_{xyz}$ and return $\vec{v}_{xyz}$.

`r3_rec_sph(v_xyz)`
Return the spherical coordinate equivalent of rectangular coordinate $\vec{v}_{xyz}$.

`r3_rec_sph_to(vi_xyz, vo_rpt)`
Save the spherical coordinate equivalent of rectangular coordinate input $\vec{v}_{xyz}$ into output $\vec{v}_{r\phi\theta}$ and return $\vec{v}_{r\phi\theta}$.

`r3_sph_rec(v_rpt)`
Return the rectangular coordinate equivalent of spherical coordinate $\vec{v}_{r\phi\theta}$.

`r3_sph_rec_to(vi_rpt, vo_xyz)`
Save the rectangular coordinate equivalent of spherical coordinate input $\vec{v}_{r\phi\theta}$ into output $\vec{v}_{xyz}$ and return $\vec{v}_{xyz}$.

`r3_cyl_sph(v_rpz)`
Return the spherical coordinate equivalent of cylindrical coordinate $\vec{v}_{r\phi z}$.

`r3_cyl_sph_to(vi_rpz, vo_rpt)`
Save the spherical coordinate equivalent of cylindrical coordinate input $\vec{v}_{r\phi z}$ into output $\vec{v}_{r\phi\theta}$ and return $\vec{v}_{r\phi\theta}$.

`r3_sph_cyl(v_rpt)`
Return the cylindrical coordinate equivalent of spherical coordinate $\vec{v}_{r\phi\theta}$.

`r3_sph_cyl_to(vi_rpt, vo_rpz)`
Save the cylindrical coordinate equivalent of spherical coordinate input $\vec{v}_{r\phi\theta}$ into output $\vec{v}_{r\phi z}$ and return $\vec{v}_{r\phi z}$.

2×2 Matrix Operations

`r22_add(M1, M2)`
Return $M_1 + M_2$.

`r22_add_to(M1, M2, Mout)`
Save $M_1 + M_2$ into M_{out} and return M_{out}.

`r22_subtract(M1, M2)`

Return $M_1 - M_2$.

`r22_subtract_to(M1, M2, Mout)`

Save $M_1 - M_2$ into M_{out} and return M_{out}.

`r22_scale(M, r)`

Return rM.

`r22_scale_to(M, r, Mout)`

Save rM into M_{out} and return M_{out}.

`r22_transpose(M)`

Return the transpose of M.

`r22_transpose_to(M, Mout)`

Save the transpose of M into M_{out} and return M_{out}.

`r22_multiply(M1, M2)`

Return $M_1 M_2$.

`r22_multiply_to(M1, M2, Mout)`

Save $M_1 M_2$ into M_{out} and return M_{out}.

`r22_transform(M, v)`

Return $M\vec{v}$.

`r22_transform_to(M, v, vout)`

Save $M\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r22_invert(M)`

Return the inverse of M. If M is singular, **`undefined`** is returned.

`r22_invert_to(M, Mout)`

Save the inverse of M into M_{out} and return M_{out}. If M is singular, M_{out} remains unaltered and **`undefined`** is returned.

`r22_encode_string(v)`

Return the string form of M.

`r22_decode_string(str)`

Return the matrix represented by **`str`**.

`r22_decode_string_to(str, Mout)`

Save the matrix represented by **`str`** into M_{out} and return M_{out}.

`r22_encode_base64(M)`
Return the Base64 string form of M.

`r22_decode_base64(enc)`
Return the matrix represented by Base64 string **`enc`**.

`r22_decode_base64_to(enc, Mout)`
Save the matrix represented by Base64 string **`enc`** into M_{out} and return M_{out}.

3×3 Matrix Operations

`r33_add(M1, M2)`
Return $M_1 + M_2$.

`r33_add_to(M1, M2, Mout)`
Save $M_1 + M_2$ into M_{out} and return M_{out}.

`r33_subtract(M1, M2)`
Return $M_1 - M_2$.

`r33_subtract_to(M1, M2, Mout)`
Save $M_1 - M_2$ into M_{out} and return M_{out}.

`r33_scale(M, r)`
Return rM.

`r33_scale_to(M, r, Mout)`
Save rM into M_{out} and return M_{out}.

`r33_transpose(M)`
Return the transpose of M.

`r33_transpose_to(M, Mout)`
Save the transpose of M into M_{out} and return M_{out}.

`r33_multiply(M1, M2)`
Return $M_1 M_2$.

`r33_multiply_to(M1, M2, Mout)`
Save $M_1 M_2$ into M_{out} and return M_{out}.

`r33_transform(M, v)`
Return $M\vec{v}$.

`r33_transform_to(M, v, vout)`

Save $M\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r33_invert(M)`

Return the inverse of M. If M is singular, undefined is returned.

`r33_invert_to(M, Mout)`

Save the inverse of M into M_{out} and return M_{out}. If M is singular, M_{out} remains unaltered and undefined is returned.

`r33_encode_string(M)`

Return the string form of M.

`r33_decode_string(str)`

Return the matrix represented by **`str`**.

`r33_decode_string_to(str, Mout)`

Save the matrix represented by **`str`** into M_{out} and return M_{out}.

`r33_encode_base64(M)`

Return the Base64 string form of M.

`r33_decode_base64(enc)`

Return the matrix represented by Base64 string **`enc`**.

`r33_decode_base64_to(enc, Mout)`

Save the matrix represented by Base64 string **`enc`** into M_{out} and return M_{out}.

4×4 Matrix Operations

`r44_add(M1, M2)`

Return $M_1 + M_2$.

`r44_add_to(M1, M2, Mout)`

Save $M_1 + M_2$ into M_{out} and return M_{out}.

`r44_subtract(M1, M2)`

Return $M_1 - M_2$.

`r44_subtract_to(M1, M2, Mout)`

Save $M_1 - M_2$ into M_{out} and return M_{out}.

`r44_scale(M, r)`

Return rM.

`r44_scale_to(M, r, Mout)`
Save rM into M_{out} and return M_{out}.

`r44_transpose(M)`
Return the transpose of M.

`r44_transpose_to(M, Mout)`
Save the transpose of M into M_{out} and return M_{out}.

`r44_multiply(M1, M2)`
Return $M_1 M_2$.

`r44_multiply_to(M1, M2, Mout)`
Save $M_1 M_2$ into M_{out} and return M_{out}.

`r44_transform(M, v)`
Return $M\vec{v}$.

`r44_transform_to(M, v, vout)`
Save $M\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`r44_invert(M)`
Return the inverse of M. If M is singular, undefined is returned.

`r44_invert_to(M, Mout)`
Save the inverse of M into M_{out} and return M_{out}. If M is singular, M_{out} remains unaltered and undefined is returned.

`r44_encode_string(M)`
Return the string form of M.

`r44_decode_string(str)`
Return the matrix represented by **`str`**.

`r44_decode_string_to(str, Mout)`
Save the matrix represented by **`str`** into M_{out} and return M_{out}.

`r44_encode_base64(M)`
Return the Base64 string form of M.

`r44_decode_base64(enc)`
Return the matrix represented by Base64 string **`enc`**.

`r44_decode_base64_to(enc, Mout)`
Save the matrix represented by Base64 string **`enc`** into M_{out} and return M_{out}.

Square Matrix Operations

`rnn_add(M1, M2)`
Return $M_1 + M_2$.

`rnn_add_to(M1, M2, Mout)`
Save $M_1 + M_2$ into M_{out} and return M_{out}.

`rnn_subtract(M1, M2)`
Return $M_1 - M_2$.

`rnn_subtract_to(M1, M2, Mout)`
Save $M_1 - M_2$ into M_{out} and return M_{out}.

`rnn_scale(M, r)`
Return rM.

`rnn_scale_to(M, r, Mout)`
Save rM into M_{out} and return M_{out}.

`rnn_transpose(M)`
Return the transpose of M.

`rnn_transpose_to(M, Mout)`
Save the transpose of M into M_{out} and return M_{out}.

`rnn_multiply(M1, M2)`
Return $M_1 M_2$.

`rnn_multiply_to(M1, M2, Mout)`
Save $M_1 M_2$ into M_{out} and return M_{out}.

`rnn_transform(M, v)`
Return $M\vec{v}$.

`rnn_transform_to(M, v, vout)`
Save $M\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rnn_invert(M)`
Return the inverse of M. If M is singular, undefined is returned.

`rnn_invert_to(M, Mout)`
Save the inverse of M into M_{out} and return M_{out}. If M is singular, M_{out} remains unaltered and undefined is returned.

`rnn_encode_string(M)`
Return the string form of M.

`rnn_decode_string(str)`
Return the matrix represented by **`str`**.

`rnn_decode_string_to(str, Mout)`
Save the matrix represented by **`str`** into M_{out} and return M_{out}.

`rnn_encode_base64(M)`
Return the Base64 string form of M.

`rnn_decode_base64(enc, n)`
Return the matrix represented by Base64 string **`enc`**.

`rnn_decode_base64_to(enc, n, Mout)`
Save the matrix represented by Base64 string **`enc`** into M_{out} and return M_{out}.

General Matrix Operations

`rmn_add(M1, M2)`
Return $M_1 + M_2$.

`rmn_add_to(M1, M2, Mout)`
Save $M_1 + M_2$ into M_{out} and return M_{out}.

`rmn_subtract(M1, M2)`
Return $M_1 - M_2$.

`rmn_subtract_to(M1, M2, Mout)`
Save $M_1 - M_2$ into M_{out} and return M_{out}.

`rmn_scale(M, r)`
Return rM.

`rmn_scale_to(M, r, Mout)`
Save rM into M_{out} and return M_{out}.

`rmn_transpose(M)`
Return the transpose of M.

`rmn_transpose_to(M, Mout)`
Save the transpose of M into M_{out} and return M_{out}.

`rmn_multiply(M1, M2)`

Return $M_1 M_2$.

`rmn_multiply_to(M1, M2, Mout)`

Save $M_1 M_2$ into M_{out} and return M_{out}.

`rmn_transform(M, v)`

Return $M\vec{v}$.

`rmn_transform_to(M, v, vout)`

Save $M\vec{v}$ into $\vec{v}_{out}$ and return $\vec{v}_{out}$.

`rmn_encode_string(M)`

Return the string form of M.

`rmn_decode_string(str)`

Return the matrix represented by **`str`**.

`rmn_decode_string_to(str, Mout)`

Save the matrix represented by **`str`** into M_{out} and return M_{out}.

`rmn_encode_base64(M)`

Return the Base64 string form of M.

`rmn_decode_base64(enc, m, n)`

Return the matrix represented by Base64 string **`enc`**.

`rmn_decode_base64_to(enc, m, n, Mout)`

Save the matrix represented by Base64 string **`enc`** into M_{out} and return M_{out}.