

Some calculator utilities for calculus III

If you found this document in a folder named "calculator" which also contains a number of files with names like "calc3.acc.v2f", then you can use TIConnect to transfer these functions directly to your calculator. If you prefer manual entry, the code for each function is included below.

First, download the "calculator" folder to your PC.

Then, start the TIConnect software, which came with your Voyage200.

Select TI GroupExplorer.

Find and open the "calculator" folder on the display.

Select the relevant files, all of which have the extension ".v2f" or ".v2i" or ".v2t".

Under the Actions menu, select Send to Device.

Follow the instructions. You'll need to connect your calculator to your computer with the USB cable that came with the calculator.

acc computes the acceleration of a moving particle whose position is given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .

Syntax: `acc(r)` or `acc([t,t^2,t^3])`

Result: `[0 2 6t]`

Code: `acc(r)`
`d(vel(r),t)`

aln computes the length of a curve given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .
For curves in 2-space, enter 0 for the third component.

Syntax: `aln(r,start,end)` or `aln([t,t^2,t^3],0,1)`

Result: 1.86302 (The calculator can't evaluate this integral exactly.)

Code: `aln(r)`
`∫(spd(r),t,a,b)`

Note: The "∫" is the integral symbol, 2nd 7.

contents is a text file that contains essentially the information in this document.

crv computes the curvature of a curve given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .

Syntax: `crv(r)` or `crv([t,t^2,t^3])`

Result: $2\sqrt{9t^4 + 9t^2 + 1} / (9t^4 + 4t^2 + 1)^{3/2}$

Code: `crv(r)`
`vl(d(utv(r),t))/(vl(vel(r)))`

curl computes the curl of a vector field.
The variables are always x, y, and z.

Syntax: `curl(v)` or `curl([x*y,y/z,z^2])`
Result: `[y/z^2 0 -x]`
Code: `curl(v)`
`[[d(v[1,3],y)- d(v[1,2],z), d(v[1,1],z)- d(v[1,3],x), d(v[1,2],x)- d(v[1,1],y)]]`
(Single brackets work as well as double brackets here.)

div computes the divergence of a vector field.
The variables are always x, y, and z.

Syntax: `div(v)` or `div([x*y,y/z,z^2])`
Result: `y+2z+1/z`
Code: `div(v)`
`d(v[1,1],x)+ d(v[1,2],y)+ d(v[1,3],z)`

dss computes dS, for surface integrals
You supply the parameters for the surface.

Syntax: `dss(r,u,v)` or `dss([x*y,x^2,y^2],x,y)`
Result: `[4x*y -2y^2 -2x^2]`
Code: `dss(r,x,y)`
`CrossP(d(r,x), d(r,y))`

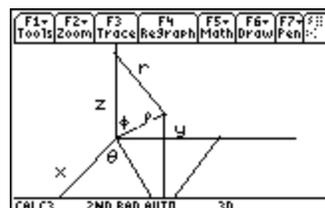
flux computes $F \cdot dS = \text{dotp}(f, \text{dss})$. You supply the variables.

Syntax: `flux(f,r,u,v)` or `flux([u^2,v^2,u*v],[u+v,u-v,u],u,v)`
Result: `u^2-2uv+v^2`
Code: `flux(f,r,x,y)`
`DotP(f,dss(r,x,y))`

grad computes the gradient of a function
The variables are always x, y, and z.

Syntax: `grad(w)` or `grad(x^2*y-x*y^2)`
Result: `[2xy-y^2 x^2-2xy 0]`
Code: `grad(f)`
`[[d(f,x), d(f,y), d(f,z)]]` (Single brackets work as well as double brackets here.)

picture displays the picture that shows
the relationships among rectangular,
cylindrical, and spherical coordinates.



pot computes potential functions for vector fields, whether they exist or not. (Computes "potential potential functions"?) The variables are always x , y , and z .

Syntax: `pot(r)` or `pot([x,y,z])`

Result: $(x^2+y^2+z^2)/2$

Code: `pot(vv)`
`potfunc(vv[1,1],vv[1,2],vv[1,3])`

potfunc a utility function called by **pot**; you can also use it on its own. The variables are always x , y , and z .

Syntax: `potfunc(x,y,z)`

Result: $(x^2+y^2+z^2)/2$

Code: `potfunc(aa,bb,cc)`
`f'(aa,x)+f'(bb,y)+f'(cc,z)-f'(f'(d(aa,y),y),x)-f'(f'(d(aa,z),z),x)-`
`f'(f'(d(bb,z),z),y)+f'(f'(f'(d(aa,y),z),z),y),x)`

spd computes the speed of a moving particle whose position is given by $\mathbf{r}(t) = [x(t),y(t),z(t)]$. The variable is always t .

Syntax: `spd(r)` or `spd([t,t^2,t^3])`

Result: $\sqrt{9t^4 + 4t^2 + 1}$

Code: `spd(r)`
`vl(vel(r))`

stretch computes $|dS| = vl(dss(r,u,v))$ for surface integrals. You supply the parameters for the surface.

Syntax: `stretch([2u^2,v^2,2u*v],u,v)`

Result: $8u^2 + 4v^2$

Code: `stretch(r,x,y)`
`vl(dss(r,x,y))`

ubv computes the length of the unit binormal vector \mathbf{B} for a moving particle whose position is given by $\mathbf{r}(t) = [x(t),y(t),z(t)]$. The variable is always t .

Syntax: `ubv(r)` or `ubv([t,t^2,t^3])`

Result: $[3t^2 - 3t \ 1] / \sqrt{9t^4 + 9t^2 + 1}$

Code: `ubv(r)`
`crossp(utv(r),unv(r))`

unv computes the length of the unit normal vector **N** for a moving particle whose position is given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .

Syntax: `unv(r) unv([t,t^2,t^3])`
Result: $[-9t^3-2t \ -9t^4+1 \ 6t^3+3t] / (\sqrt{9t^4 + 4t^2 + 1}) \sqrt{9t^4 + 9t^2 + 1}$
Code: `unv(r)`
`utv(utv(r))`

utv computes the length of the unit tangent vector **T** for a moving particle whose position is given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .

Syntax: `utv(r) utv([t,t^2,t^3])`
Result: $[1 \ 2t \ 3t^2] / \sqrt{9t^4 + 4t^2 + 1}$
Code: `utv(r)`
`vel(r)/vl(vel(r))`

vel computes the velocity of a moving particle whose position is given by $\mathbf{r}(t) = [x(t), y(t), z(t)]$.
The variable is always t .

Syntax: `vel(r) vel([t,t^2,t^3])`
Result: $[1 \ 2t \ 3t^2]$
Code: `unv(r)`
`d(r,t)`

vl computes the length of a vector in 3-space.
Avoids the complex conjugation of the built-in function **norm**

Syntax: `vl([3,4,t])`
Result: $\sqrt{25+t^2}$
Code: `vl(r)`
 $\sqrt{r[1,1]^2+r[1,2]^2+r[1,3]^2}$

vsolve solves equations involving vectors
You can use this in conjunction with integration operations to solve projectile motion problems. It handles vectors with any (finite) number of components, or matrices, constrained by the amount of available memory in your calculator.

Syntax: vsolve(vector equation, list of variables enclosed in braces)

Syntax: vsolve([a^2,b,3]=[b,2,a]+[1,1,1],{a,b})

Result: a=2 and b=3

Syntax: vsolve(a*[100,<b]=[500,300],{a,b})

Result: a=5.83095 and b=.54042

Code: vsolve(veqn,ls)

Func

Local vv,i,n,e

breakup(veqn)→vv

dim(ls)→n

"solve(vv,{ls[1]" →e

For i,2,n

e&" ,ls["&string(i)&"]" →e

EndFor

e&"})" →e

expr(e)

EndFunc

breakup is a utility program called by **vsolve**. It has no independent applications.

Code: breakup(veqn)

Func

Local a,m,n,i,j

dim(veqn)[1]→m

dim(veqn)[2]→n

true→a

For i,1,m

For j,1,n

a and veqn[i,j]→a

EndFor

EndFor

EndFunc
