Using Vectors to Represent Lines and Find Distances

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

Introduction

OBJECTIVE: Visualize and interpret the use of vectors to represent lines in the plane.

Do you remember how you first learned about the equations of lines in a plane? In this module, you will gain insight into why it is to your advantage to interpret lines in the plane using vectors.

Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then shut down *Maple* and start it up again.

Part I: Interpreting Lines Using Their Vector Definition

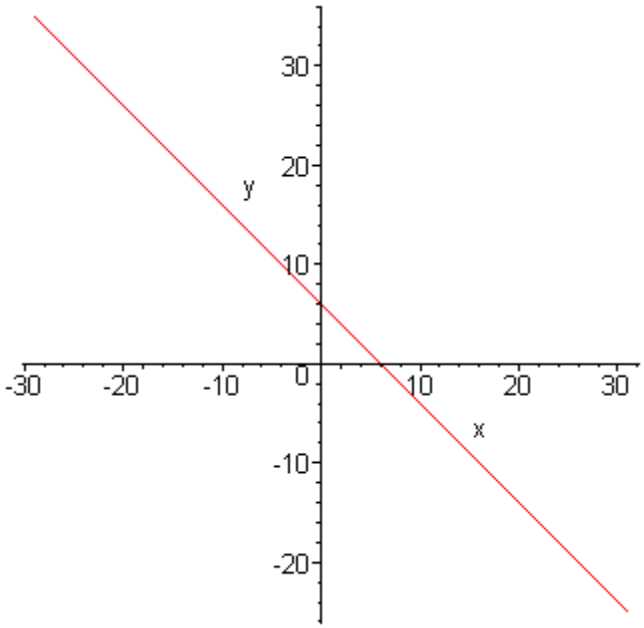
You can construct straight lines in two dimensions using the vector definition. Given two points on the line, you will first determine the direction of the line and then write any other point on the line as the position vector to that point plus a multiple (t) of the vector in the direction of the line. We plot points along the line corresponding to different values of the parameter t. Our example uses points [*1, 5*] and [*4, 2*] and we plot for values of t from -10 to 10.

```
> restart:
```

```
> o := [0,0]:
  p1:= [1,5]:
  p2:= [4,2]:
  dir:= p2-p1:
  print(`The direction of the line is`, dir);
  print(`[x,y]=`, expand(p1-t*dir));
  plot([(p1[1]-o[1])+t*dir[1],p1[2]-o[2]+t*dir[2], t=-10..10],labels=["x","y"]);
```

$$\text{The direction of the line is, } [3, -3]$$

$$[x,y]=, [-3t + 1, 3t + 5]$$



You Try It: Part I

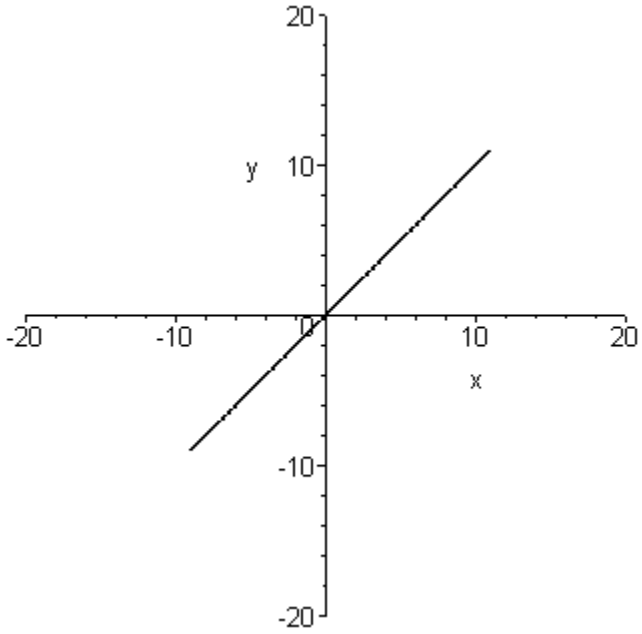
Write and plot the line connecting the points (-23, -5) and (10, 12) in parametric form

You need to identify and enter the new points for *p1* and *p2* on the line and re-execute the input cell.

```
> o := [0,0]:
  p1:= [1,1]:
  p2:= [2,2]:
  dir:= p2-p1:
  print(`The direction of the line is`, dir);
  print(`[x,y]=`, expand(p1-t*dir));
  plot1:=plot([(p1[1]-o[1])+t*dir[1],p1[2]-o[2]+t*dir[2], t=-10..10], labels=["x","y"], co
  view=[-20..20, -20..20], thickness=2):
  print(plot1);
```

The direction of the line is, [1, 1]

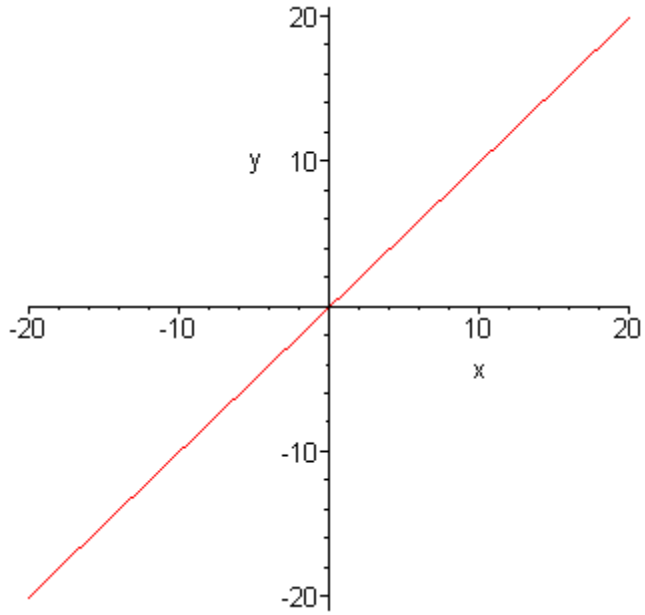
$[x,y]=, [-t+1, -t+1]$

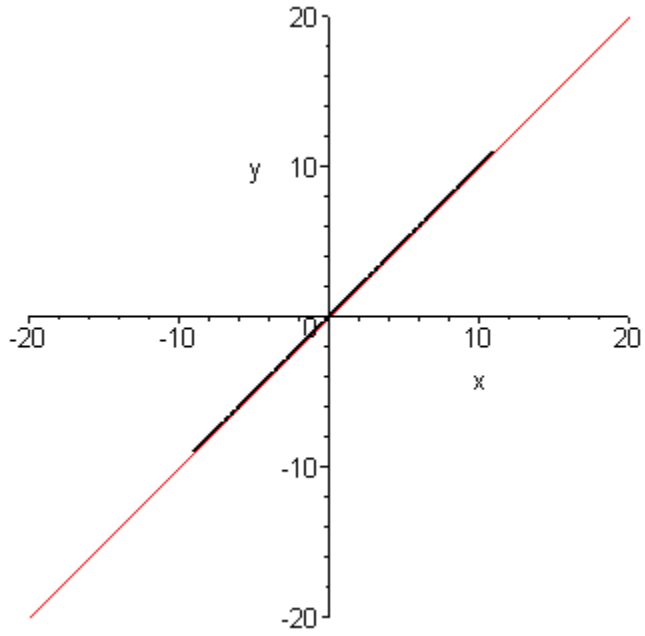


For the preceding function, match the parametric plot with the standard Cartesian form and plot.

You need to solve for *y* as a function of *x* and put that into the expression for ***f.*** Replace the expression in red with the appropriate expression.

```
> f:=x:
  pf:=plot(f(x)-.1,x=-20..20, labels=["x","y"],color=red):
  print(pf);
  print(plots[display]({pf,plot1}));
```





Part II: Different Parametric Forms for the Same Line

Write the equation of the line $y = 3x - 5$ in parametric form.

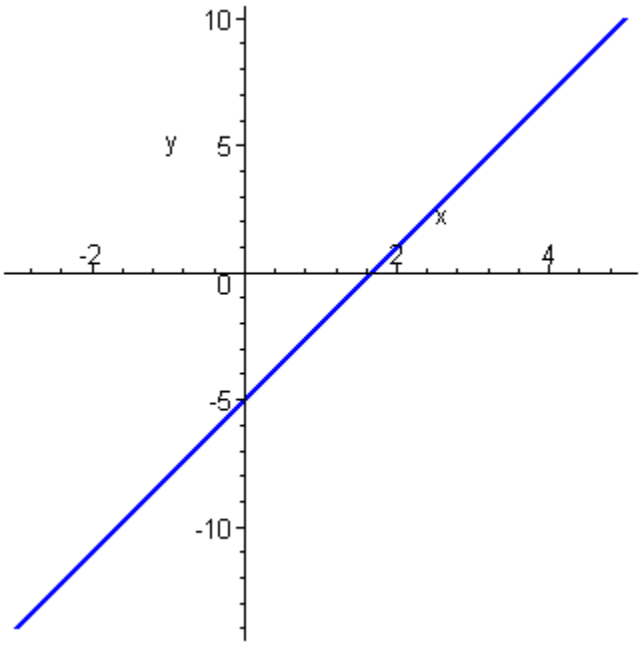
You may be accustomed to seeing equations of lines in the standard slope, intercept form. That form is a unique representation. In contrast, the parametric form can appear in different forms; however, the set of points represented is the same.

If we take the direction vector from the slope to be [*1, 3*] and use the *y*-intercept, [*0, 5*] as our given point on the line, we get the following.

> **o:= [0, 0]:**

```
dir := [1, 3]:
p1 := [0, -5]:
print(`[x,y] =`, expand(p1+dir*t));
plot1:=plot([(p1[1]-o[1])+t*dir[1],p1[2]-o[2]+t*dir[2], t=-3..5], color=[blue],thickness=3, lal
["x","y"]):
print(plot1);
```

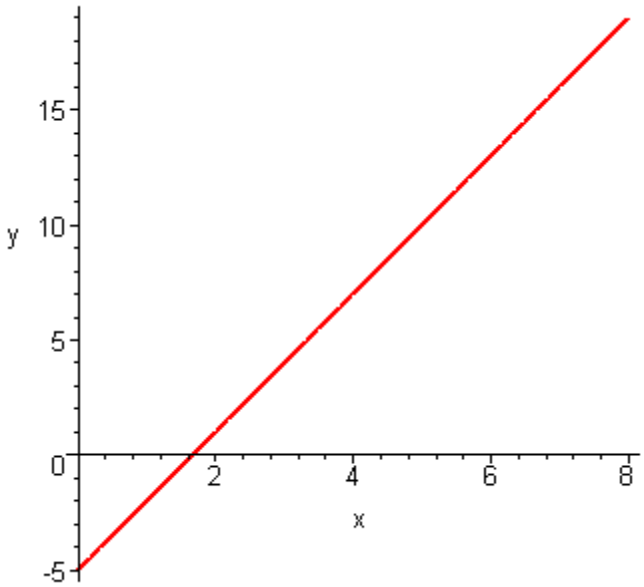
$$[x,y] =, [t, 3\,t - 5]$$



Had we used a different point on the line, for example, (*2, 1*), and written the slope as (*2, 6*), the equation would look a bit different, although it would produce the same line, even though we are plotting only sements of that line.

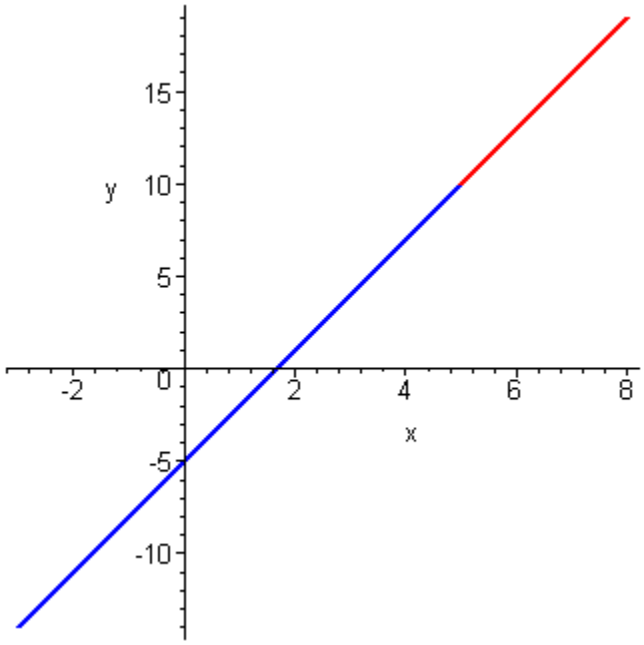
```
> dir2:=[2,6]:
  q2:=[2,1]:
  eq:=(q2-o) + dir*t:
  print(`[x,y]=`, eq);
  plot2:=plot([(q2[1]-o[1])+t*dir2[1],q2[2]-o[2]+t*dir2[2], t=-1..3], thickness=[3],labels=
  ["x","y"]):
  print(plot2);
```

$$[x,y]=, [2, 1] + t\,[1, 3]$$



We can look at these graphs together.

> **print(plots[display]({plot1,plot2}));**

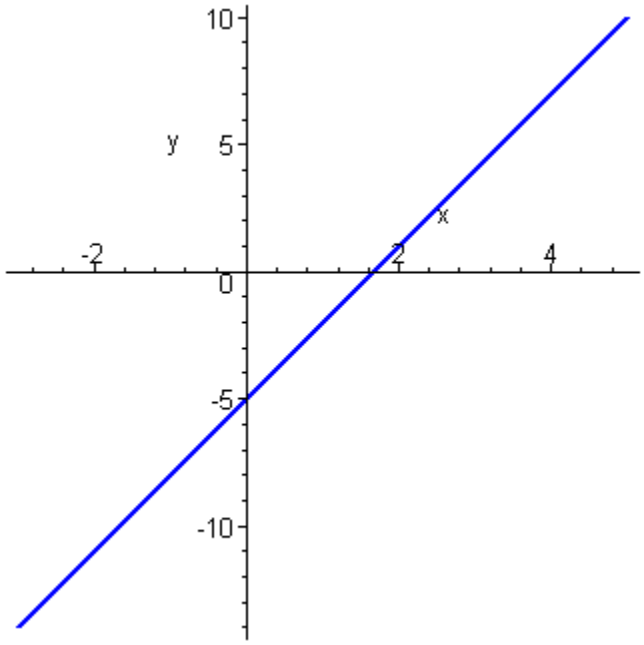


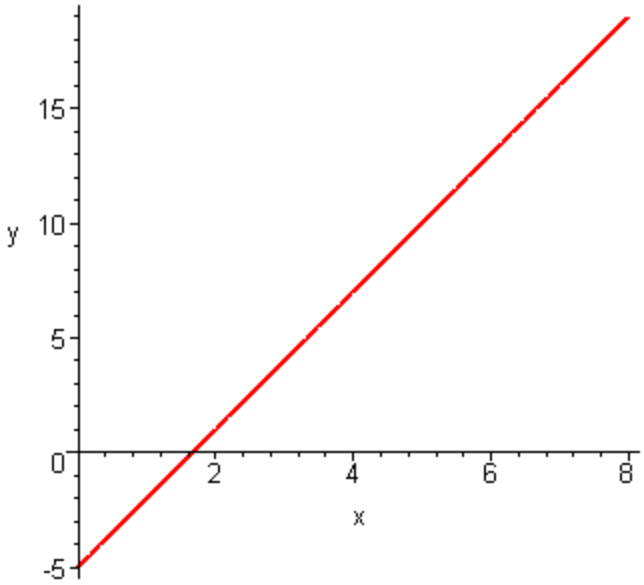
Note that the parameter *t* is not the same in the two equations. However, all the points lie on the same line.

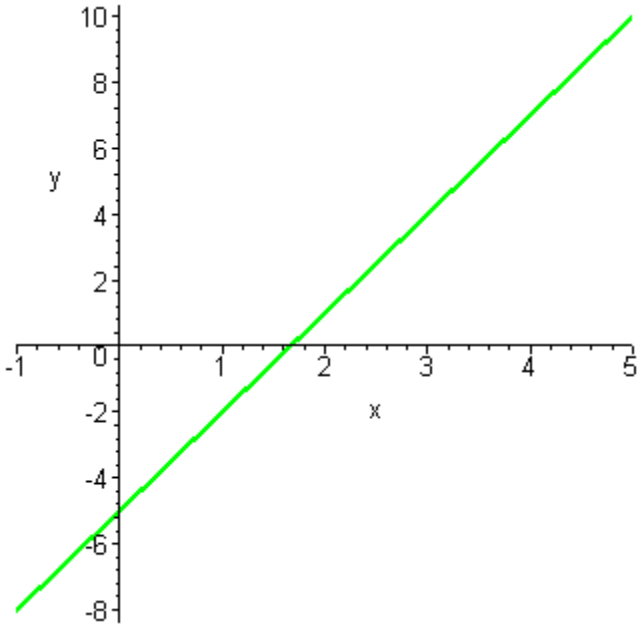
For the preceding function, match the parametric plot with the standard Cartesian form and plot.

> **f:=3*x-5:**
pf:=plot(f(x), x=-1..5, labels=["x","y"], colour=green, thickness=3):

print(plot1); print(plot2); print(pf);







Do they match?

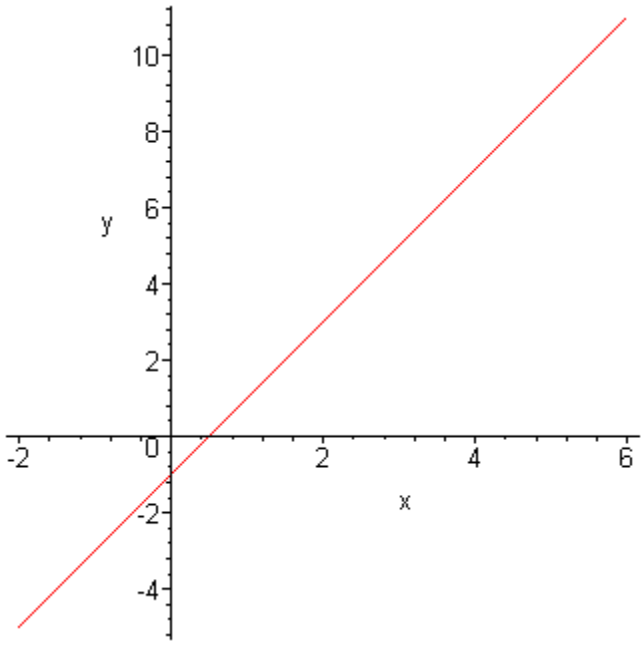
You Try It: Part II

Write the line $4x + 5y = 12$ in parametric form.

You need to identify a point on the line and a vector in the direction of the line. Substitute these for **dir** and **p1**, and re-execute the input cell.

```
> dir:=[1,2]:
  p1:=[1,1]:
  print(`[x,y]=`, expand((p1-o)+t*(dir)));
  plot1:= plot([(p1[1]-o[1])+t*dir[1],p1[2]-o[2]+t*dir[2], t=-3..5],labels=["x","y"]):
  print(plot1);
```

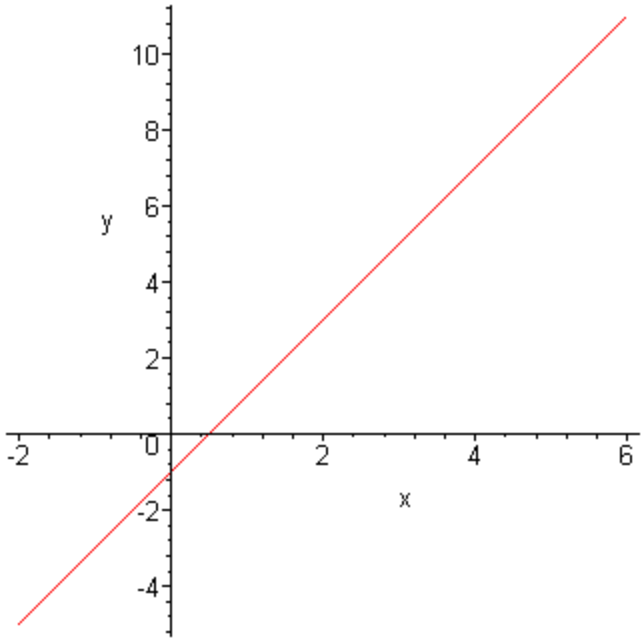
$$[x,y]=, [t + 1, 2\,t + 1]$$

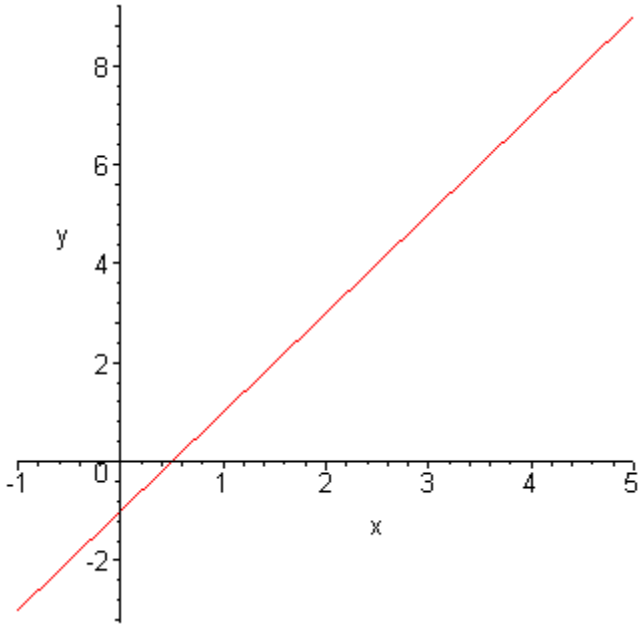


For the preceding function, match the parametric plot with the standard Cartesian form and plot.

You need to solve for *y* as a function of *x* and put that into the expression for *f*. Replace the *f* with the appropriate expression.

```
> f:=2*x-1:
  pf:=plot(f(x), x=-1..5, labels=["x","y"]):
  print(plot1); print(pf);
```





Part III: Finding the Distance from a Point to a Line

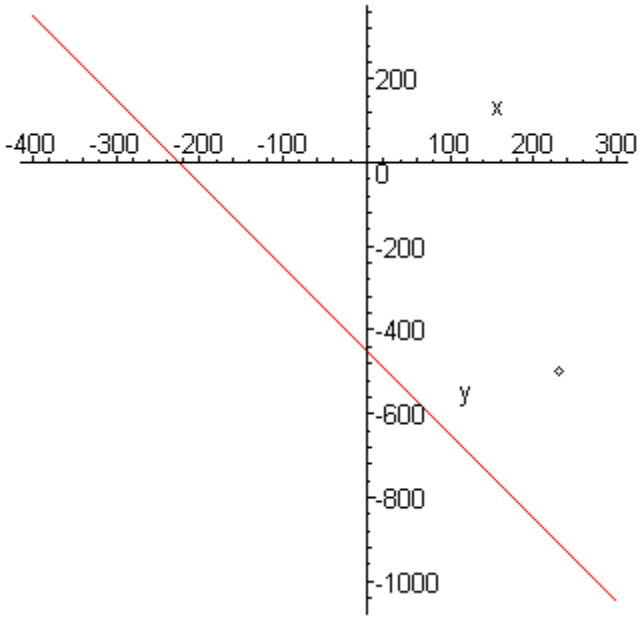
Suppose you are driving on a long, straight road on a flat planar region of desert land. You have a grid-map of the region and have been told that there is a radiation site in the at the point [30, -500]. You started at the point [-400, 350] and are traveling in a path that is always in the direction of the vector

[1, -2]. You are worried about how close to the radiation site will you pass.

Let's begin by visualizing the problem.

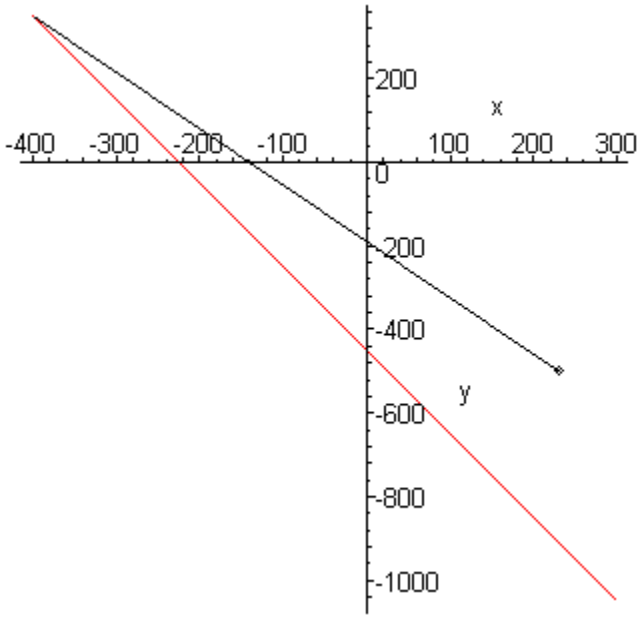
```
> o:=[0,0]:
  site:=[230,-500]:
  start:=[-400,350]:
  dir:=[1,-2]:
  print(`[x,y]=`, expand((start-o)+t*dir));
  pointA:= plots[pointplot](site):
  plotA:= plot([(start[1]-o[1])+t*dir[1],(start[2]-o[2])+t*dir[2], t=0..700],labels=["x","y"]):
  print(plots[display]({pointA,plotA}));
```

$$[x,y]=, [t-400, -2t+350]$$



Consider the vector from the starting point to the radiation site.

```
> vect:=site-start:
  vp:=plots[pointplot]([site, start], style=line,labels=["x","y"]):
  print(plots[display]({pointA,plotA,vp}));
```



Now, use the dot product to find the projection of this vector onto a direction perpendicular to the line. That will represent the shortest distance between the radiation site and the road.

Finding the perpendicular to a vector in two-space is very easy. Think of how perpendicular lines have slopes that are negative reciprocals of each other. You will find the perpendicular to the line by simply reversing the x and y components of the direction vector and negating one of them.

```
> perpdir:=[-dir[2], dir[1]];
```

$$perpdir := [2, 1]$$

Then, to find the projection of the vector connecting the starting point with the radiation site onto this perpendicular direction use, the dot product of those two vectors and divide by the magnitude of the perpendicular vector.

```
> distance:=evalf(linalg[dotprod](vect, perpdir)/sqrt(linalg[dotprod](perpdir, perpdir)));
```

$$distance := 183.3575741$$

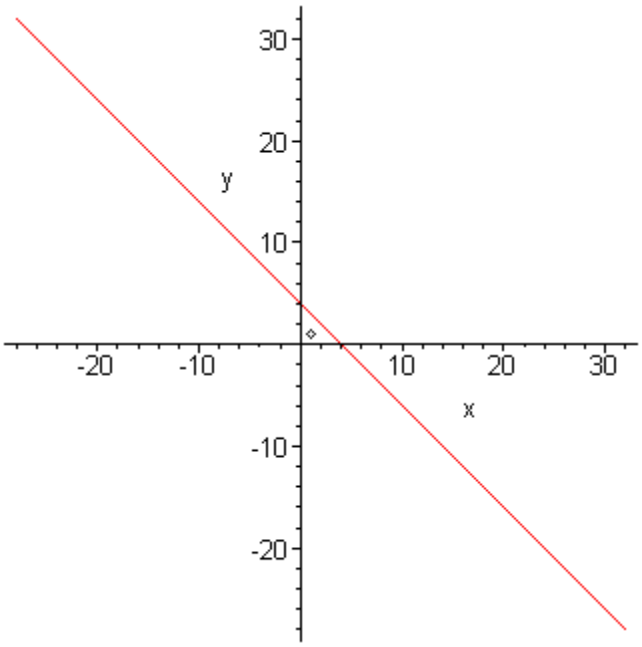
You Try It: Part III

Find the shortest distance from the line $4x + 5y = 12$ to the point $[-2, 3]$.

In the following set of commands, to put in the new site, select a point on the line, and identify the direction of the line. To do this, just replace the appropriate expressions.

```
> o:=[0,0]:
  site:=[1,1]:
  start:=[2,2]:
  dir:=[-3,3]:
  vect:=site-start:
  t:=`t`:
  print(`[x,y]=`, expand((start-o)+t*dir));
  pointA:= plots[pointplot](site):
  plotA:= plot([(start[1]-o[1])+t*dir[1],(start[2]-o[2])+t*dir[2], t=-10..10],labels=["x","y"]):
  print(plots[display]({pointA,plotA}));
  perpdir:=[-dir[2], dir[1]]:
  distance:=evalf(linalg[dotprod](vect, perpdir)/sqrt(linalg[dotprod](perpdir, perpdir))):
  print(`the shortest distance from the point to the line is`, distance);
```

$$[x,y]=, [-3\,t + 2, 3\,t + 2]$$



the shortest distance from the point to the line is, 1.414213562

>