JuliaPlotExamples

April 8, 2021

1 Creating vector plot using Julia

based on stackoverflow question

1.1 Packages

The Plots.jl package will allow us to create plots using various backends

There are many backends available for plotting, such as - gr() (default backend good for creating fast plots) - plotly() (good for interactive plots) - pyplot() (a highly customizable python library for plotting)

Here we use the plotly backend to create the plot.

[1]:
```
using Plots;
```

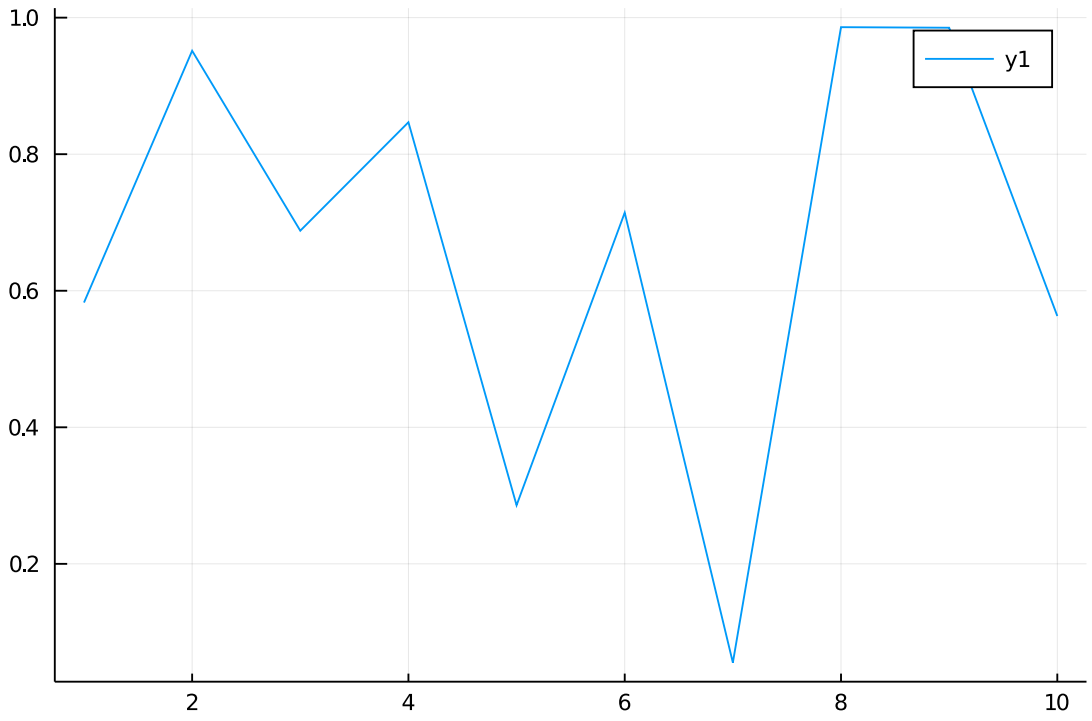
The SpecialFunctions is a package containinng many special mathematical functions listed here

[2]:
```
using SpecialFunctions;
```

Some Simple Plot Commands

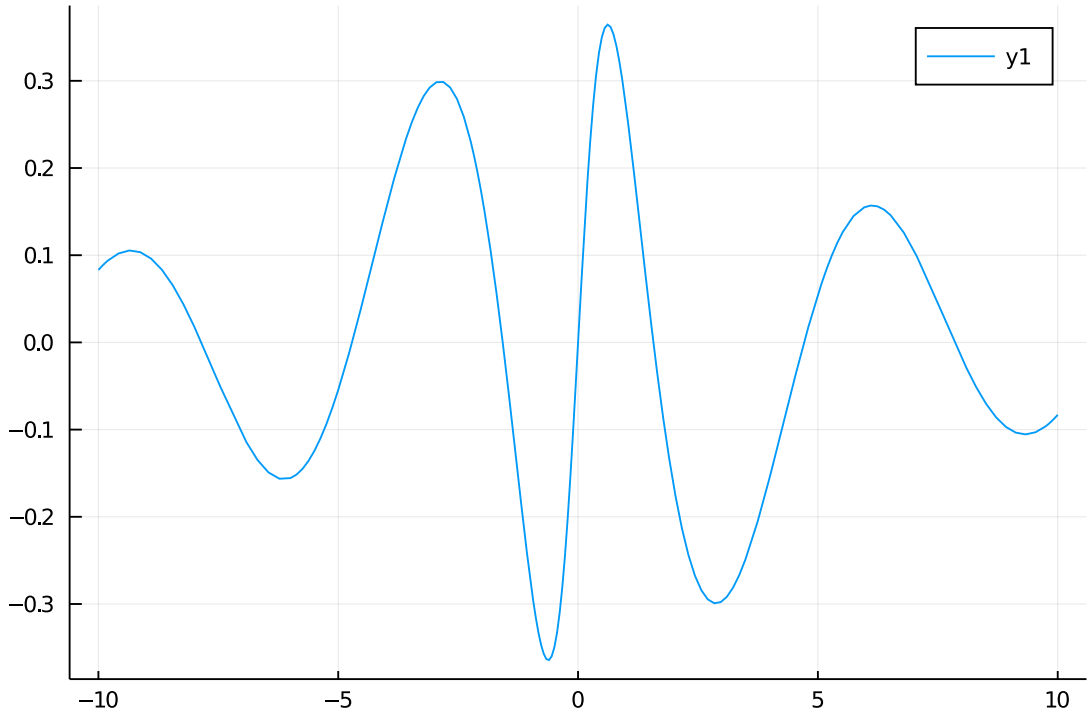
[3]:
```
x = 1:10
y = rand(10)
plot(x,y)
```

[3]:



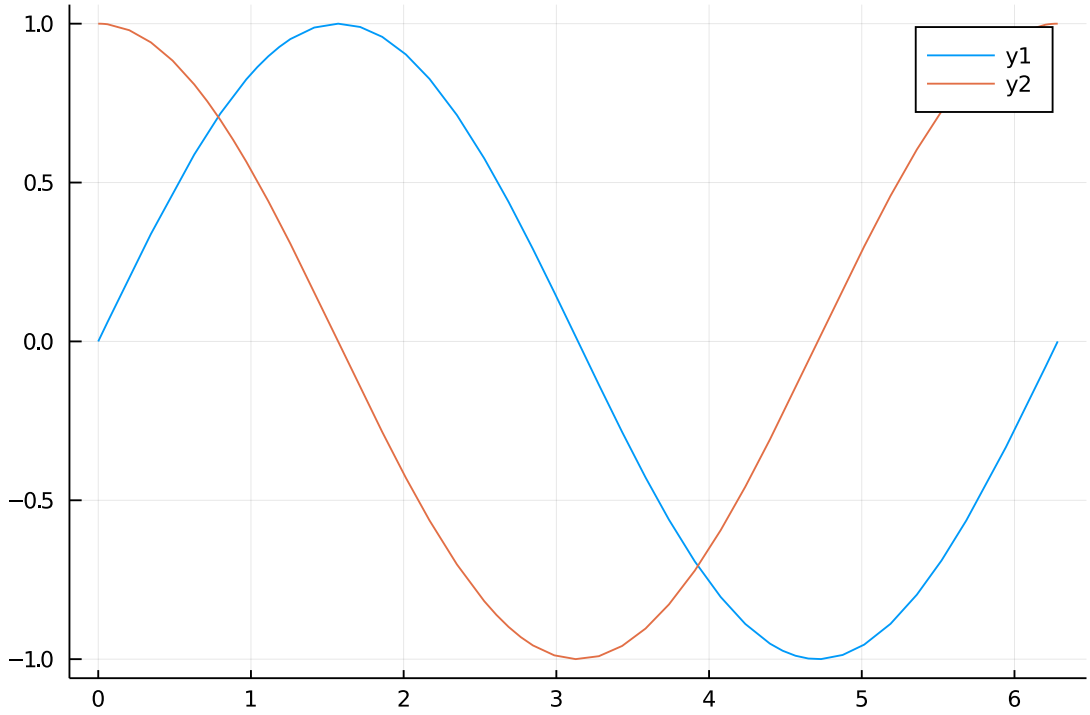
[4]:
```
f(x) = cos(x) * x/(1 + x^2)
plot(f, -10,10)
```

[4]:



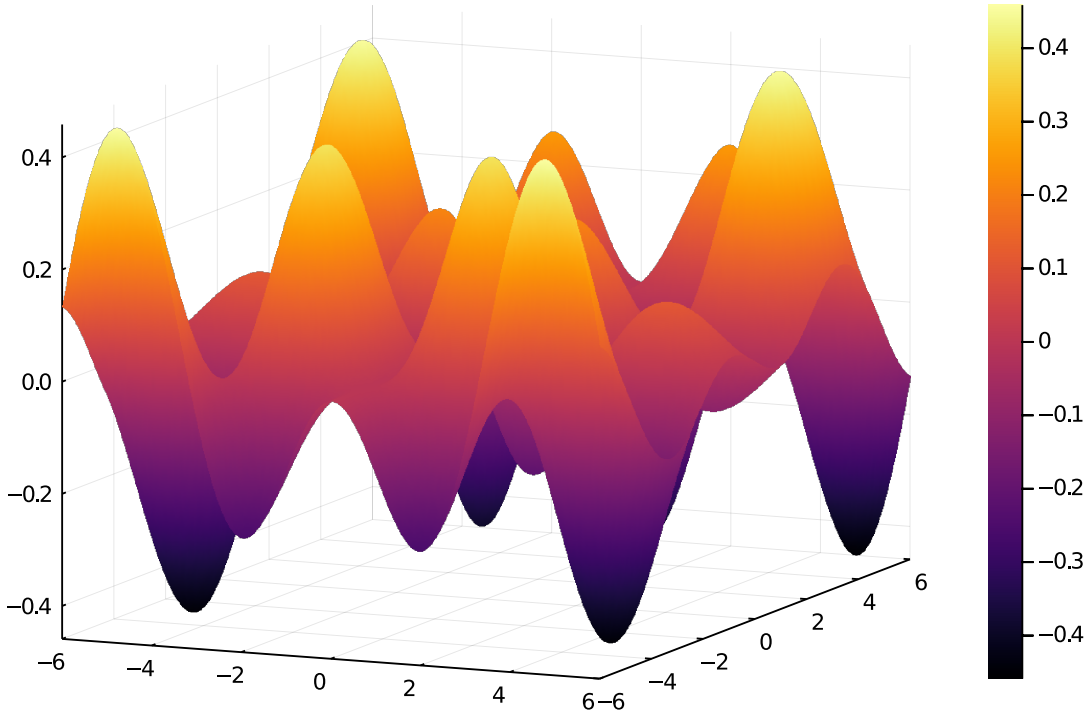
[5]:
```
plot([sin, cos], 0, 2pi)
```

[5]:



[6]:
```
f(x,y) = sin(x)*cos(y)*x * y / (x^2 + y^2 + 1);
x = collect(Float16, range(-6,length=200,stop=6));
y = collect(Float16, range(-6,length=200,stop=6));
surface(x,y,f.(x',y))
```

[6]:



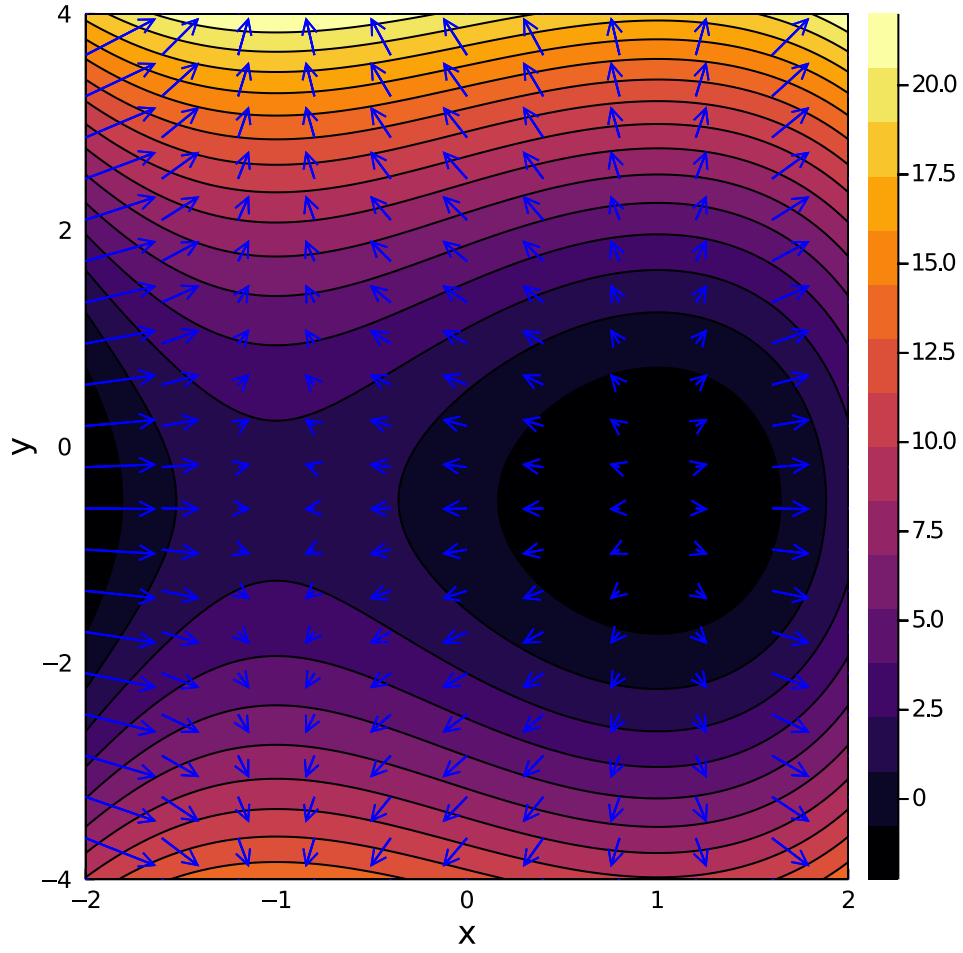
1.2 Sample plot based on stack exchange question

[7]:
```
# plotted using gr() backend
gr(size=(500,500))
# create sample points along x and y directions
X = range(-2, stop=2, length=100)
Y = range(-4, stop=4, length=100)
# the function to be plotted
f(x, y) = x^3 - 3x + y^2+y
# here we create the contour plot
# (without the vectors)
contour(X, Y, f,fill=true)
# number of vector points
# in the x and y direction
nqx=11
nqy=22
# again create sample points in x and y direction
x = range(-2, stop=2, length=nqx)
y = range(-4, stop=4, length=nqy)
# the gradient of the function used in the contour
# here we divide by 25 to adjust the magnitude of the
# vectors
df(x, y) = [3x^2 - 3; 2y+1] / 25
```

```
# quiver! is used to place the vectors at each
# samplepoint x and y
quiver!(repeat(x,nqx), vec(repeat(y',nqy)), quiver=df, c=:blue)

xlims!(-2, 2)
ylims!(-4, 4)
plot!(xlab="x", ylab="y")
```

[7]:



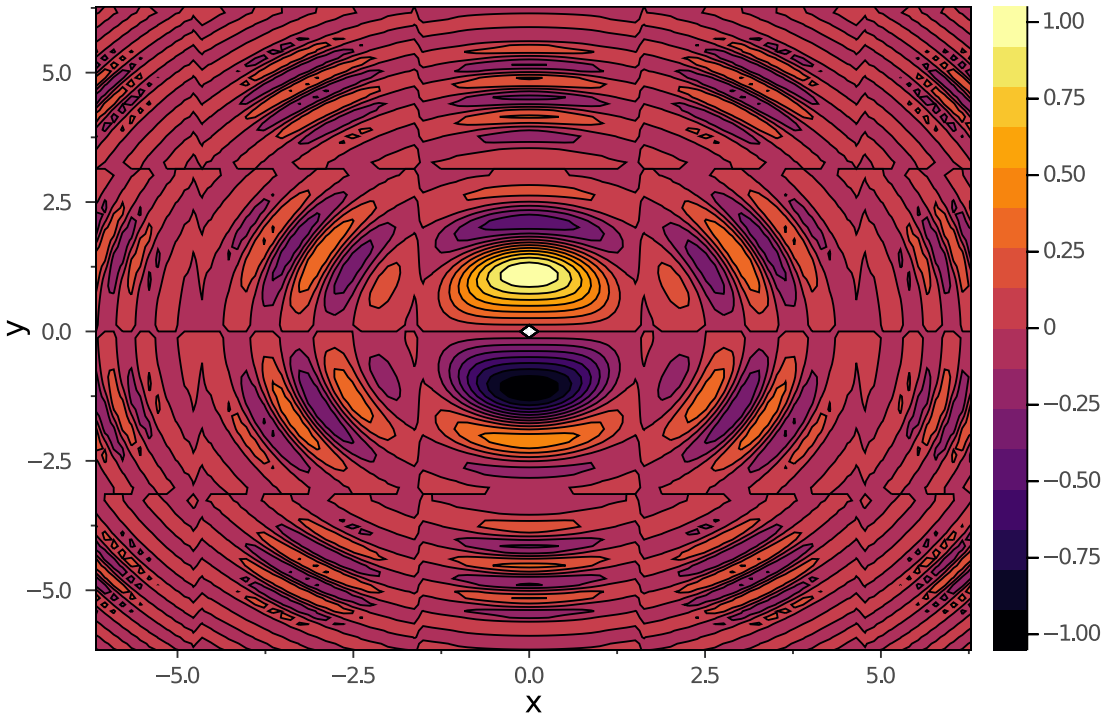
1.3 Sample Plot Similar to one found in PlotlyJS Documentation

1.3.1 We recreate plot plot done in PlotlyJS using gr() to show different approaches

[8]:
```
theme(:ggplot2)
gr(size=(600,400))
# create sample points along x and y directions
# This time we create the points using a for loop
```

```
# so that we can set different values for each point
x = [-2*pi + 4*pi*i/100 for i in 1:100]
y = [-2*pi + 4*pi*i/100 for i in 1:100]
f(x, y) = sin(y) * cos(x) * sin(x*x+y*y)/log(x*x+y*y+1)
contour(x, y, f,fill=true)
plot!(xlab="x", ylab="y")
```

[8]:



1.4 Plotting the same function using two different methods

1.4.1 Plot $\cos x \sin y$ as a 2d contour plot and an interactive 3d plot

[9]:
```
using Plots;
theme(:dark)
gr(size=(600,600));
x=range(-2*pi, stop=2*pi, length=250);
y=range(-2*pi, stop=2*pi, length=250);
```

[10]:
```
# create 2 nxn arrays of points to sample from
xx = reshape([xi for xi in x for yj in y],  length(y), length(x));
yy = reshape([yj for xi in x for yj in y],  length(y), length(x));
```

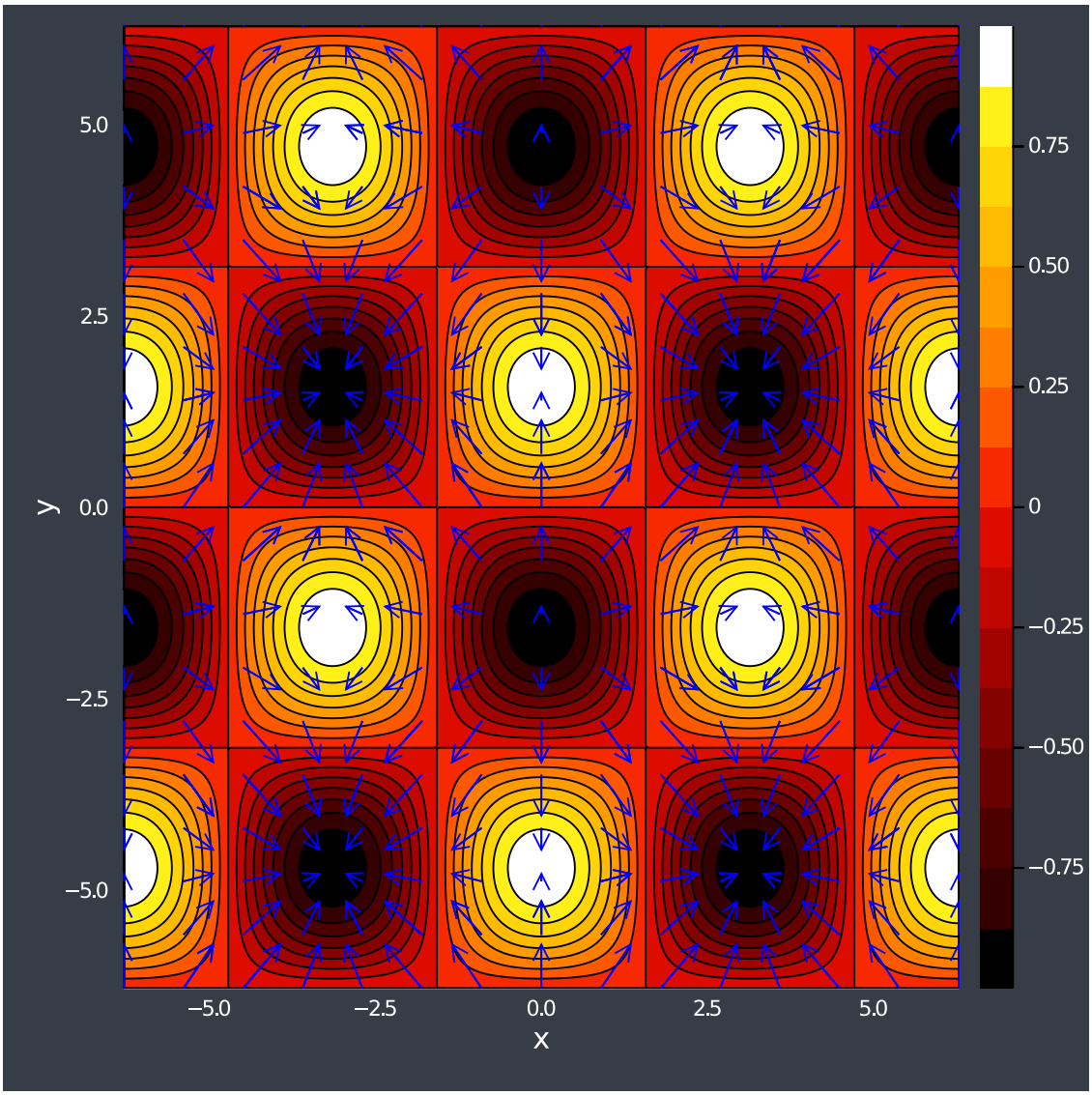
now pass the sample falues to the funtion - Note that the .(x) notation is called broadcasting and it is used to indicate to Julia that the function sin has to be computed for each element of x - Where code written in Python and Matlab will see

drastic improvements in speed from vectorizing code. This is not necessary in Julia. This is because Python and Matlab are writen in C so vectorizing code will reduce the amount of time spent passing code to the lower level language. But Julia is written in Julia (similar to how C is written in C) so vectorization is not neccessary. - sin.(xx) can be thought of as mapping the sin function onto all of the values in the array xx

[11]:
```
f(x,y)=cos.(x).*sin.(y);
contour(x,y,f,fill=true)
nqx=15
nqy=19
x = range(-2*pi, stop=2*pi, length=nqx)
y = range(-2*pi, stop=2*pi, length=nqy)
df(x, y) = [sin(x);cos(y)]/1.75
quiver!(repeat(x,nqx), vec(repeat(y',nqy)), quiver=df, c=:blue)

xlims!(-2*pi, 2*pi)
ylims!(-2*pi, 2*pi)
plot!(xlab="x", ylab="y")
```

[11]:



1.4.2 Plot $\cos x \sin y$ as a 2d contour plot and an interactive 3d plot

Here we switch from using gr() to using plotly. This is to demonstrate plotly's ability to make 3d interactive plots

This makes them extremely easy to add plots to a website using simple html code

<iframe width="100%" height="450px" frameborder="0" scrolling="no" src="/foo/bar/sincos.html"></iframe>

[12]:
```
plotly();
theme(:dark)
```

Info: For saving to png with the Plotly backend PlotlyBase has to be installed.

```
@ Plots /home/ulj/.julia/packages/Plots/uCh2y/src/backends.jl:372
```

[13]:
```
zz = cos.(xx).*sin.(yy);
plot3d(xx,yy,zz, label=:none, st = :surface)
plot!(xlab="x", ylab="y", zlab="cos(x)*sin(y)")
```

1.4.3 Creating Plots using special functions

- Here we plot the zeta function (from the specialfunctions pacage) on the complex plane

[14]:
```
x=range(-1, stop=2, length=100)
y=range(-20, stop=20, length=100)
xx = reshape([xi for xi in x for yj in y],  length(y), length(x));
yy = reshape([yj for xi in x for yj in y],  length(y), length(x));
# We again use broadcasting (as explained above)
# We first broadcast the built in function complex across
# the x/y coordinates creating a complex number with the
# x representing the real numbers and the y the imaginary
# We then broadcast the zeta function across this array
# and finally the aboslute value
zz=abs.(zeta.(complex.(xx,yy)))

plot3d(xx,yy,zz,
    label=:none,
    st = :surface,
    xlim = (-1, 2),
    ylim = (-20, 20),
    zlim = (0, 5),
    xlab = "Real",
    ylab = "Imaginary",
    zlabel = "Zeta",
    size=(640, 480),
    autosize = false,
    width= 500,
    height = 250,
    c=cgrad([:orange, :blue], [0.0125, 0.025, 0.05,0.1],scale=:exp))
```

[15]:
```
theme(:dao)
plotly(size=(600,600));
x=range(-1, stop=2, length=100)
y=range(-20, stop=20, length=100)
f(x,y)=abs.(zeta.(complex.(x,y)));
data=contour(x,y,f,fill=true)
plot(data,
    xlab = "Real",
    ylab = "Imaginary",
```

```
    xlim = (-1, 1),
    ylim = (-20, 20),
    title="the zeta function",
    zlim = (0, 5))
```