1. (a) By definition, $\vec{\varphi}(t)$ is a flow for $(W^t, \mathbb{R}^2)$ if $\frac{d}{dt}\vec{\varphi}(t) = \vec{F}(\vec{\varphi}(t))$, so let's check that:

   i. $\vec{\varphi}(t) = (-e^{-2t}, \sqrt{3}e^{-t})$, then

   $$\frac{d}{dt}\vec{\varphi}(t) = (2e^{-2t}, -\sqrt{3}e^{-t})$$

   and

   $$\vec{F}(\vec{\varphi}(t)) = (-e^{-2t}+(\sqrt{3}e^{-t})^2, -\sqrt{3}e^{-t}) = (-e^{-2t}+3e^{-2t}, -\sqrt{3}e^{-t}) = (2e^{-2t}, -\sqrt{3}e^{-t})$$

   Thus, $\frac{d}{dt}\vec{\varphi}(t) = \vec{F}(\vec{\varphi}(t))$. It is a flow.

   ii. $\vec{\varphi}(t) = (-e^{-4t}, \sqrt{3}e^{-2t})$, then

   $$\frac{d}{dt}\vec{\varphi}(t) = (4e^{-4t}, -2\sqrt{3}e^{-2t})$$

   and

   $$\vec{F}(\vec{\varphi}(t)) = (-e^{-4t}+(\sqrt{3}e^{-2t})^2, -\sqrt{3}e^{-2t}) = (-e^{-4t}+3e^{-4t}, -\sqrt{3}e^{-2t}) = (2e^{-4t}, -\sqrt{3}e^{-2t})$$

   Thus, $\frac{d}{dt}\vec{\varphi}(t) \neq \vec{F}(\vec{\varphi}(t))$. It is not a flow.

   iii. $\vec{\varphi}(t) = (\frac{4}{3}e^t - \frac{1}{3}e^{-2t}, e^{-t})$, then

   $$\frac{d}{dt}\vec{\varphi}(t) = (\frac{4}{3}e^t + \frac{2}{3}e^{-2t}, -e^{-t})$$

   and

   $$\vec{F}(\vec{\varphi}(t)) = (\frac{4}{3}e^t-\frac{1}{3}e^{-2t}+(e^{-t})^2, -e^{-t}) = (\frac{4}{3}e^t-\frac{1}{3}e^{-2t}+e^{-2t}, -e^{-t}) = (\frac{4}{3}e^t+\frac{2}{3}e^{-2t}, -e^{-t})$$

   Thus, $\frac{d}{dt}\vec{\varphi}(t) = \vec{F}(\vec{\varphi}(t))$. It is a flow.

   iv. $\vec{\varphi}(t) = (e^t - e^{-2t}, e^{-t})$, then

   $$\frac{d}{dt}\vec{\varphi}(t) = (e^t + 2e^{-2t}, -e^{-t})$$

   and

   $$\vec{F}(\vec{\varphi}(t)) = (e^t - e^{-2t} + (e^{-t})^2, -e^{-t}) = (e^t - e^{-2t} + e^{-2t}, -e^{-t}) = (e^t, -e^{-t})$$

   Thus, $\frac{d}{dt}\vec{\varphi}(t) \neq \vec{F}(\vec{\varphi}(t))$. It is not a flow.

   (b) We want to find constant flows $\vec{\varphi}$ of $\vec{F}$ such that $\vec{\varphi}(t_1) = \vec{\varphi}(t_2)$ for all $t_1, t_2 \in \mathbb{R}$. This is equivalent as $\vec{\varphi}\,'(t) = \vec{0}$ for all $t \in \mathbb{R}$. Suppose $\vec{\varphi}(t) = (t_1, t_2)$. Then,

   $$\vec{\varphi}\,'(t) = \vec{F}(\vec{\varphi}(t)) = (t_1 + t_2^2, -t_1) = (0, 0)$$

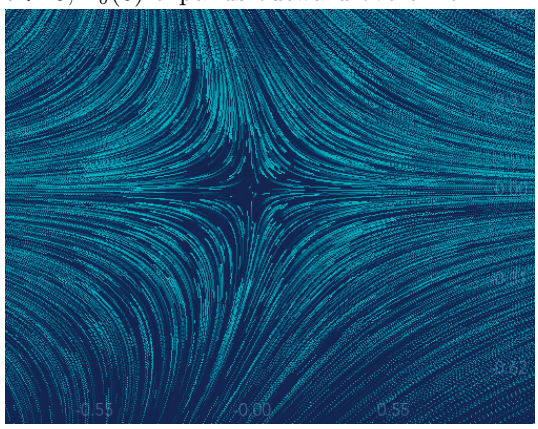
   since $\vec{\varphi}(t)$ is a flow if $\vec{\varphi}\,'(t) = \vec{F}(\vec{\varphi}(t))$, which means

   $$t_1 + t_2^2 = 0, -t_1 = 0 \implies t_2^2 = 0 \implies t_1, t_2 = 0$$

   Then, all the constant flows for $\vec{F}$ is the trivial flow $\vec{\varphi}(t) = \vec{0}$

   (c) The fixed points of $(W^t, \mathbb{R}^2)$ are $\vec{x} \in \mathbb{R}^2$ such that $W^t(\vec{x}) = \vec{x}$ for all $t \in \mathbb{R}$, and $W^t(\vec{x}) = \vec{\varphi}(t)$ by definition. Then, we want to find such $\vec{x}$ such that $\vec{\varphi}(t) = \vec{x}$ for all $t \in \mathbb{R}$. These means the constant flows are the fixed points. From 1*b*), we know the constant flow is $\vec{0}$, and we know from in class exercise, it is unstable.
   Below is a visual representation of the vector field $\vec{F}$. One can clearly see that for any

$\delta > 0, B_\delta(\vec{0})$ expands outward overtime.



(d) We want to find $\vec{\varphi}(t)$ such that $\frac{d}{dt}\vec{\varphi}(t) = \vec{F}(\vec{\varphi}(t))$ and $\vec{\varphi}(t) = (1,0)$ for some $t \in \mathbb{R}$. With computer program's help, we find the general solution for $\vec{\varphi}(t) = (c_2e^t - \frac{1}{3}c_1^2e^{-2t}, c_1e^{-t})$ for some $c_1, c_2 \in \mathbb{R}$. Next, want to see for what c_1, c_2 that $\vec{\varphi}(t) = (c_2e^t - \frac{1}{3}c_1^2e^{-2t}, c_1e^{-t}) = (1,0)$ for some $t \in \mathbb{R}$, and we get

$$c_2e^t - \frac{1}{3}c_1^2e^{-2t} = 1$$

and

$$c_1e^{-t} = 0 \implies c_1 = 0$$

so

$$c_2e^t = 1 \implies t = \ln\frac{1}{c_2} \implies c_2 > 0$$

Then, $\vec{\varphi}(t) = (ce^t, 0)$ for some $c > 0 \in \mathbb{R}$ such that it is a flow that passes through $(1,0)$ at $t = \ln\frac{1}{c}$.

(e) Fix flows $\vec{\varphi}_1$ and $\vec{\varphi}_2$ from part (d) such that $\vec{\varphi}_1 = (c_1e^t, 0)$ and $\vec{\varphi}_2 = (c_2e^t, 0)$ for some $c_1, c_2 > 0 \in \mathbb{R}$. We want to show that they are time shifts of each other, i.e., $\vec{\varphi}_1(t) = \vec{\varphi}_2(t + t_0)$ for some t_0.
Pick $t_0 = \ln\frac{c_1}{c_2}$, t_0 is defined as $\frac{c_1}{c_2} > 0$. Then $\vec{\varphi}_2(t + t_0) = (c_2e^{t+t_0}, 0) = (c_2e^te_0^t, 0) = (c_2e^te^{\ln\frac{c_1}{c_2}}, 0) = (c_2e^t\frac{c_1}{c_2}, 0) = (c_1e^t, 0) = \vec{\varphi}_1(t)$, as needed.

2. (a) Suppose $x \in X$ is a periodic point for (T, X), so there exists an $i \neq 0$ so that $T^i x = x$. We want to show that x is a periodic point for (W^t, X), i.e., there exists a $t \neq 0$ so that $W^t(x) = x$.
Pick $t = i \neq 0$. Then, $W^t(x) = W^i(x) = W^1(W^1(...(W^1(x))))$ for i times of W^1 by definition of (W^t, X) being a continuous dynamical system, and $W^1(W^1(...(W^1(x)))) = T(T(...(T(x))))$ for i times of T as well. Thus, $W^t(x) = W^i(x) = W^1(W^1(...(W^1(x)))) = T(T(...(T(x)))) = T^i(x) = x$, as needed.

   (b) Suppose $x \in X$ is a periodic point for (W^t, X), i.e., there exists a $t \neq 0$ so that $W^t(x) = x$. x might not be a periodic point for (T, X) if $t \not\equiv 0 \bmod 1$, as composition of $i \neq 0 \in \mathbb{Z}$ times of x under T will not give us $T^i(x) = x$, since T is defined as a time-1 map of W^t.

   (c) Assume x is a point of period at least 2 for (T, X), so $T^i(x) = x$ where $i \geq 2$. Notice that $i \in \mathbb{N}$ as it is the number of times for composition of T. We want to show that there are infinitely many periodic points for (T, X) of the same period i.
From $2a$), we know that x is also a periodic point for W^t where $W^i(x) = x$.
We also know $W^q(x) = y \neq x$ for $0 < q < 2, q \in \mathbb{R}$ by $i \geq 2$ being the period of x (the minimum value such that $T^i(x) = x$).
Claim: There are infinitely many $W^q(x) = y$ for $0 < q < 2, q \in \mathbb{R}$, and they are periodic points for T and has a period of i.
We know there are infinite many y for $q \in (0, 2)$. Check if any of such y is a periodic point for T with period i. $T^i(y) = T^i(W^q(x)) = W^i(W^q(x)) = W^q(W^i(x)) = W^q(x) = y$ by definition of (W^t, X) being a continuous dynamical system, $T^i(x) = W^i(x) = x$, and $W^q(x) = y$. This completes our proof of the claim.

   (d) We want to show that the logistic map $T : [0, 1] \to [0, 1]$ defined by $x \mapsto \frac{19}{6}x(1 - x)$ has exactly two points of period 2. Using computer algebra system, we find $x = \frac{10}{19}, \frac{15}{19}$ to be the solutions to $T^2(x) = \frac{19}{6}(\frac{19}{6}x(1 - x))(1 - \frac{19}{6}x(1 - x)) = x$, and $T(\frac{10}{19}) = \frac{15}{19}, T(\frac{15}{19}) = \frac{10}{19} \neq x$ respectively.

   (e) No, $2d$)'s logistic map T with $r = \frac{19}{6}$ is a such discrete dynamical systems that is not a time-1 maps to continuous dynamical systems. Suppose for the sake of contradiction that it is. We know from $2d$) it only has exactly two points of period 2, but from $2c$) we know it should have infinitely many points of period 2. This is a contradiction.

3. The function $G : \mathbb{R}^n \to \mathbb{R}^m$ is called *affine* if there exists a matrix A and a vector $\vec{p}$ so that $G(\vec{x}) = A\vec{x} + \vec{p}$ for all $\vec{x}$.

   Let $F : \mathbb{R}^n \to \mathbb{R}^m$. A *first-order approximation* to F at the point $\vec{w} \in \mathbb{R}^n$ is an affine function $L : \mathbb{R}^n \to \mathbb{R}^m$ satisfying

$$\lim_{\|\vec{x}\|\to 0} \frac{F(\vec{w}+\vec{x}) - L(\vec{w}+\vec{x})}{\|\vec{x}\|} = 0.$$

   (a) Let $F : \mathbb{R} \to \mathbb{R}$ be defined by $x \mapsto x^2$. We want to find a first-order approximation, $L_0 : \mathbb{R} \to \mathbb{R}$ to F at 0, where $L_0(x) = Ax + p$ for some 1 by 1 matrix $A = \begin{bmatrix} a \end{bmatrix}$ and $a, p \in \mathbb{R}$ such that $\lim_{x\to 0} \frac{F(0+x) - L_0(0+x)}{x} = 0$, then

$$\lim_{x\to 0} \frac{F(0+x) - L_0(0+x)}{x} = \lim_{x\to 0} \frac{x^2 - Ax - p}{x} = \lim_{x\to 0} \frac{x^2 - ax - p}{x}$$

$$= \lim_{x\to 0}(x - a - \frac{p}{x}) = -a - \lim_{x\to 0} \frac{p}{x} = 0$$

$$\implies a = -\lim_{x\to 0} \frac{p}{x} \implies p = -ax$$

   Pick $a = 0$, we get $L_0(x) = Ax + p = \begin{bmatrix} a \end{bmatrix} x + p = 0$.

   Similarly, we want to find a first-order approximation, $L_2 : \mathbb{R} \to \mathbb{R}$ to F at 2, where $L_2(x) = Ax + p$ for some 1 by 1 matrix $A = \begin{bmatrix} a \end{bmatrix}$ and $a, p \in \mathbb{R}$ s.t. $\lim_{x\to 0} \frac{F(2+x) - L_2(2+x)}{x} = 0$, then

$$\lim_{x\to 0} \frac{F(2+x) - L_2(2+x)}{x} = \lim_{x\to 0} \frac{(2+x)^2 - A(2+x) - p}{x}$$

$$= \lim_{x\to 0} \frac{4 + 4x + x^2 - 2a - ax - p}{x} = \lim_{x\to 0}(4 + x - a + \frac{4 - 2a - p}{x})$$

$$= 4 - a + \lim_{x\to 0} \frac{4 - 2a - p}{x} = 0$$

$$\implies 4 - a = -\lim_{x\to 0} \frac{4 - 2a - p}{x} \implies -(4-a)x = (4 - 2a - p)$$

   Pick $a = 4$, then $p = -4$ and we get $L_0(x) = Ax + p = \begin{bmatrix} a \end{bmatrix} x + p = 4x - 4$.

   (b) Let $F : \mathbb{R}^2 \to \mathbb{R}^2$ be defined by $\begin{bmatrix} x \\ y \end{bmatrix} \mapsto T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$ where $T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Define $L_{\vec{w}}$ to be the first-order approximation of F at the point $\vec{w}$. We want to find $L_{\vec{w}}(\vec{x}) = A\vec{x} + \vec{p}$ for some 2 by 2 matrix A and $\vec{p} \in \mathbb{R}^2$ s.t. $\lim_{\|\vec{x}\|\to 0} \frac{F(\vec{w}+\vec{x}) - L_{\vec{w}}(\vec{w}+\vec{x})}{\|\vec{x}\|} = 0$. Then,

$$\lim_{\|\vec{x}\|\to 0} \frac{F(\vec{w}+\vec{x}) - L_{\vec{w}}(\vec{w}+\vec{x})}{\|\vec{x}\|} = \lim_{\|\vec{x}\|\to 0} \frac{T(\vec{w}+\vec{x}) - A(\vec{w}+\vec{x}) - \vec{p}}{\|\vec{x}\|}$$

$$= \lim_{\|\vec{x}\|\to 0} \frac{T(\vec{w}) + T(\vec{x}) - A(\vec{w}) - A(\vec{x}) - \vec{p}}{\|\vec{x}\|} = 0$$

$$\implies T(\vec{w}) + T(\vec{x}) - A(\vec{w}) - A(\vec{x}) - \vec{p} = 0 \implies T(\vec{w}) - A(\vec{w}) + (T(\vec{x}) - A(\vec{x})) = \vec{p}$$

   If we pick $\vec{p} = \vec{0}$, then $T(\vec{x}) = A(\vec{x})$ for all $\vec{x} \in \mathbb{R} \implies T(\vec{w}) = A(\vec{w})$, so $T = A$ works. Thus, $L_{\vec{w}}(\vec{x}) = A\vec{x} + \vec{p} = T(\vec{x}) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$.

   (c) Let $L : \mathbb{R}^n \to \mathbb{R}^m$ be an affine function, so there exists a matrix A and a vector $\vec{p}$ so that $L(\vec{x}) = A\vec{x} + \vec{p}$ for all $\vec{x}$. let $L_{\vec{w}}$ be the first-order approximation to L at $\vec{w}$, so it is also an affine function $L_{\vec{w}} : \mathbb{R}^n \to \mathbb{R}^m$ satisfying

$$\lim_{\|\vec{x}\|\to 0} \frac{L(\vec{w}+\vec{x}) - L_{\vec{w}}(\vec{w}+\vec{x})}{\|\vec{x}\|} = 0.$$

We want to show that $L = L_{\vec{w}}$ regardless of $\vec{w}$.
Fix $\vec{w}$. Suppose $L_{\vec{w}}(\vec{x}) = B\vec{x} + \vec{q}$ for some matrix B and a vector $\vec{q} \in \mathbb{R}^m$ for all $\vec{x}$, then

$$\lim_{\|\vec{x}\|\to 0} \frac{L(\vec{w}+\vec{x}) - L_{\vec{w}}(\vec{w}+\vec{x})}{\|\vec{x}\|} = 0$$

$$\implies \lim_{\|\vec{x}\|\to 0} L(\vec{w}+\vec{x}) = \lim_{\|\vec{x}\|\to 0} L_{\vec{w}}(\vec{w}+\vec{x})$$

$$\implies \lim_{\|\vec{x}\|\to 0} (A(\vec{w}+\vec{x}) + \vec{p}) = \lim_{\|\vec{x}\|\to 0} (B(\vec{w}+\vec{x}) + \vec{q})$$

$$\implies L(\vec{w}+\vec{x}) = L_{\vec{w}}(\vec{w}+\vec{x}) \ \forall \vec{x} \implies L = L_{\vec{w}}$$

as needed.

(d) Assume $(W^t, \mathbb{R}^n)$ is a continuous dynamical system with velocities given by $V(\vec{x}) = A\vec{x}$ for some matrix A. We want to show that the point $\vec{x} \in \mathbb{R}^n$ is stable under W^t if and only if the point $\vec{0}$ is stable under W^t. From 3*c*), we know that a first order approximation of V at any point $\vec{x}$ is the same as V since V is given by the matrix A. Then, calculating the eigenvalues λ of the first order approximation matrix tells us the stability at any $\vec{x}$ by solving $\det(A - \lambda I)$, so $\vec{x}$ is stable under W^t if and only if point $\vec{0}$ is stable under W^t as they have same first order approximation with the same eigenvalues.

(e) Let $F(x,y) = (f_x(x,y), f_y(x,y)) = (x + y^2, -y)$
We know $F'(x,y) = \begin{bmatrix} \frac{\partial f_x}{\partial x}(x,y) & \frac{\partial f_x}{\partial y}(x,y) \\ \frac{\partial f_y}{\partial x}(x,y) & \frac{\partial f_y}{\partial y}(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 2y \\ 0 & -1 \end{bmatrix}$
Then, a first order approximation of F at $\vec{w}_1 = (1,1)$ is

$$L_1(\vec{w}_1 + \vec{x}) = F'(\vec{w}_1)(\vec{x}) + F(\vec{w}_1) = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

Similarly, a first order approximation of F at $\vec{w}_2 = (0,0)$ is

$$L_2(\vec{w}_2 + \vec{x}) = F'(\vec{w}_2)(\vec{x}) + F(\vec{w}_2) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(f) Let $(W^t, \mathbb{R}^2)$ be the continuous dynamical system that flows along $F(x,y) = (x+y^2, -y)$. We know the eigenvalues of the first order linear approximation of F at $(1,1)$ and $(1,0)$ (computed in 3*e*)) are $\lambda_1 = 1, \lambda_2 = -1$ for $(1,1)$ and $(0,0)$. Since both have a positive real eigenvalue, we know they are both unstable as the pull force that makes them unstable increases in the direction of eigenvectors for eigenvalue 1 within the ϵ neighborhood of $(1,1)$ and $(0,0)$ while flowing along with F.

4. Stability/instability describes how points behave under a dynamical system. Let's take a moment to think about how *volumes/areas* behave.

   For this problem, you may use any facts you know about the determinant without justification (so long as they're true...).

   (a) Let $A = [\vec{a}_1|\cdots|\vec{a}_n]$ be a matrix with columns $\vec{a}_1, \ldots, \vec{a}_n$, i.e. $A = \begin{bmatrix} a_{11} & a_{21} & \ldots & a_{n1} \\ a_{12} & a_{22} & \ldots & a_{n2} \\ \ldots & \ldots & \ldots & \ldots \\ a_{1n} & a_{2n} & \ldots & a_{nn} \end{bmatrix}$ where $\vec{a}_i = \begin{bmatrix} a_{i1} \\ a_{i2} \\ \ldots \\ a_{in} \end{bmatrix}$.

   Let E_i be the identity matrix with the ith column replaced with $\vec{a}_i$. We want to show that

   $$\mathrm{Tr}(A) = \sum \det(E_i).$$

   We know $\mathrm{Tr}(A) = a_{11} + a_{22} + \cdots + a_{nn}$.

   On the other hand, $det(E_i) = det(\begin{bmatrix} 1 & \ldots & a_{i1} & \ldots & 0 \\ 0 & \ldots & a_{i2} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & a_{in} & \ldots & 1 \end{bmatrix}) = a_{ii}$, so $\sum_{i=1}^{n} det(E_i) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$.
   Thus, $\mathrm{Tr}(A) = \sum \det(E_i)$.

   (b) Let $(W^t, \mathbb{R}^2)$ be the continuous dynamical system which flows vectors along the vector field given by $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

   The limit definition of the derivative $\frac{\partial W^t}{\partial t}$ at time $t = 0$ is

   $$\frac{\partial W^t}{\partial t}|_{t=0}(\vec{x}) = \lim_{t \to 0} \frac{W^t(\vec{x}) - W^0(\vec{x})}{t} = A\vec{x}$$

   as the derivative of W^t at $t = 0$ is the tangent vector to the flow of starting at $\vec{x}$ after $t = 0$ seconds.

   (c) The first-order approximation to W^t with respect to time at $t = 0$ is

   $$L_t(\vec{x}) = \frac{\partial W^t}{\partial t}|_{t=0}(\vec{x}) * t + W^0(\vec{x}) = (A\vec{x}) * t + \vec{x}$$

   by 4*b*).

   (d) We want estimate the volume of the image of the unit cube after flowing for ε seconds. We know the first order approximation of W^t w.r.t. $t = 0$ for $t = \varepsilon$ seconds from 4*c*) is

   $$L_\varepsilon(\vec{x}) = \frac{\partial W^t}{\partial t}|_{t=0}(\vec{x}) * \varepsilon + W^0(\vec{x}) = (A\vec{x}) * \varepsilon + \vec{x}$$

   Since the unit cube volume is the same as $\det(I)$ where I is the identity matrix, we know that the image of the unit cube volume using the first order approximation of W^t w.r.t. $t = 0$ will be $\det(\varepsilon * A + I)$ as $\varepsilon * A$ is the change in velocities of area after ε seconds and we start with I.

   (e) Building up from 4*d*), we know $\det(t * A + I)$ is the area of the image of unit cube after flowing for t seconds, and

   $$\begin{aligned} \det(t * A + I) &= \det(\begin{bmatrix} at + 1 & bt \\ ct & dt + 1 \end{bmatrix}) \\ &= (at + 1)(dt + 1) - (bt)(ct) = (ad - bc)t^2 + (a + d)t \end{aligned} \tag{1}$$

Then, the instantaneous rate of change of volume (area) with respect to time for W^t is

$$\frac{\partial}{\partial t}((ad-bc)t^2+(a+d)t) = 2t(ad-bc)+(a+d) = 2t*\det(A)+(a+d)$$

by taking derverative of (1) w.r.t time.
Thus, at time $t=0$, we get the instantaneous rate of change of area w.r.t time for W^t is $a+d$.

(f) We want to show that if we drop a dab of ink of area α near the origin and let it flow for via W^t for ε seconds, the area of the resulting blob of ink will be approximately $\alpha(1+\varepsilon\,\mathrm{Tr}(A))$.
We know that the area of the resulting blob of ink, S = initial area of the blob of ink + (rate of change of volume at time = 0)∗time∗initial area of the blob of ink = α + (rate of change of volume at time = 0) $*\,\varepsilon*\alpha$.
From 4e), we know that the instantaneous rate of change of volume at $t=0$ is $(a+d)$.
Thus,

$$S = \alpha+(a+d)*\varepsilon*\alpha = \alpha(1+\varepsilon Tr(A))$$

as needed.

(g) The term incompressible is warranted.
If we have an incompressible vector field, where its divergence is zero, from 4a) − e), we know $Tr(J(F)\big|_{(x_0,y_0)}) = 0$ where $J(F)\big|_{(x_0,y_0)}$ is used in the first-order approximation to F at (x_0,y_0) for all (x_0,y_0), just like how the matrix A is used in previous questions.
Then, by 4f), we know if we drop a blob of ink with area α at any (x_0,y_0), its resulting area after flowing with N^t for ε seconds is $\alpha(1+\varepsilon\,\mathrm{Tr}(A)) = \alpha$. This means every area at any point after flowing with the continuous dynamical system which follows with an incompressible vector field does not change the initial area; N^t does not compress the area at all. This gives the name "incompressible vector field" as N^t flows points along the vector field, and it flows points to itself, which are incompressible.

Homework 3

Do the programing part of Homework 3 in this notebook. Predefined are function *stubs*. That is, the name of the function and a basic body is predefined. You need to modify the code to fulfil the requirements of the homework.

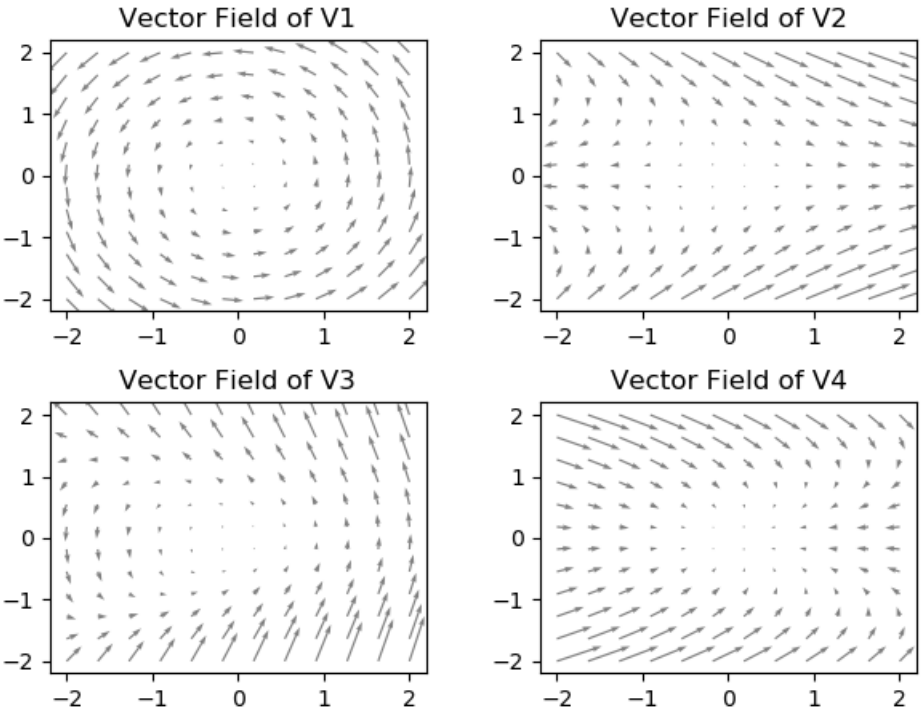
In [6]:
```
# import numpy and matplotlib
import numpy as np
import matplotlib.pylab as plt
from collections import Counter
# We give the matplotlib instruction twice, because firefox sometimes gets upset if we don't.
# note these `%`-commands are not actually Python commands. They are Jupyter-notebook-specific co
%matplotlib notebook
%matplotlib notebook
```

In [7]:
```
def plot_vectorfield(ax, F, extents=[0,1,0,1], N=20):
    """Plot the vector field determined by the function F(x,y) on the
    matplotlib "axis" `ax`.
    """
    xs,ys = np.linspace(extents[0], extents[1], N), np.linspace(extents[2], extents[3], N)
    X, Y = np.meshgrid(xs, ys)
    ax.quiver(X, Y, *F(X, Y), color="gray")
    return ax

def plot_streams(ax, F, extents=[0,1,0,1], N=20):
    """Plot the vector field determined by the function F(x,y) on the
    matplotlib "axis" `ax`.
    """
    xs,ys = np.linspace(extents[0], extents[1], N), np.linspace(extents[2], extents[3], N)
    X, Y = np.meshgrid(xs, ys)
    ax.streamplot(X, Y, *F(X, Y), color="gray")
    return ax
```

In [8]:
```
def V1(X, Y):
    return -Y, X

def V2(X, Y):
    return X + Y*Y, -Y

def V3(X, Y):
    return -Y, X + Y*Y

def V4(X, Y):
    return -X + Y*Y, -Y
```

In [9]:
```
#
# Make a single plot with the vector fields for V1,...,V4 as separate subplots
#

# (2,2) returns a figure and a 2x2 array of "axes"
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(2, 2)
# pad=2.0 makes our plots more spaced out so that the titles will fit
fig.tight_layout(pad=2.0)

#
# Plot your vectorfields here
#
plot_vectorfield(ax1, V1, [-2,2,-2,2], 12)
ax1.set_title("Vector Field of V1")
plot_vectorfield(ax2, V2, [-2,2,-2,2], 12)
ax2.set_title("Vector Field of V2")
plot_vectorfield(ax3, V3, [-2,2,-2,2], 12)
ax3.set_title("Vector Field of V3")
plot_vectorfield(ax4, V4, [-2,2,-2,2], 12)
ax4.set_title("Vector Field of V4")
```

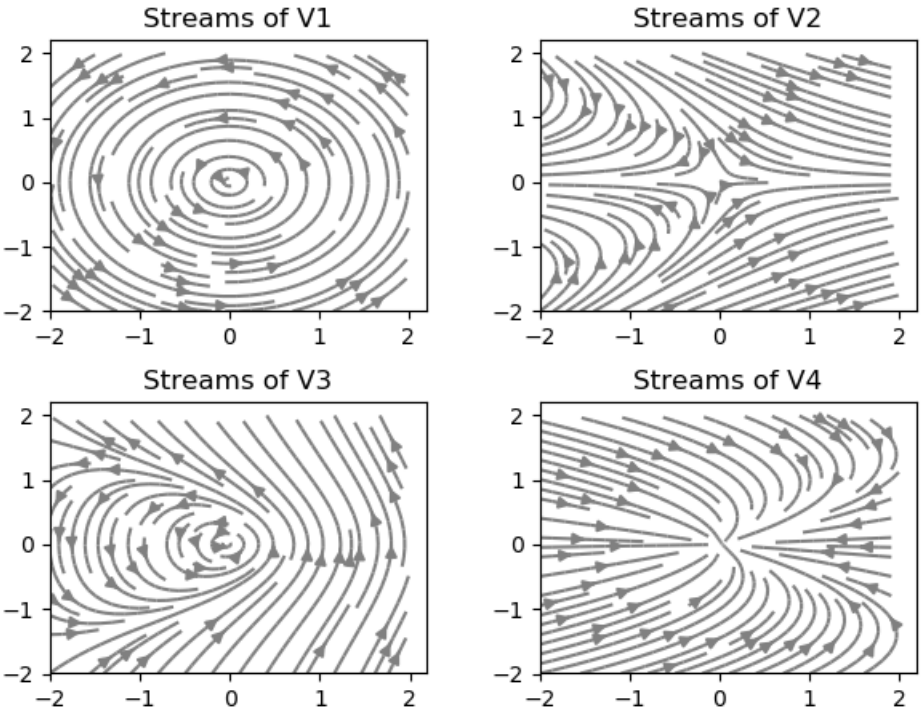
Figure 1



Out[9]: Text(0.5, 1.0, 'Vector Field of V4')

In [10]:
```
# Bouns: Make a single plot with the streams for V1,...,V4 as separate subplots

# (2,2) returns a figure and a 2x2 array of "axes"
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(2, 2)
# pad=2.0 makes our plots more spaced out so that the titles will fit
fig.tight_layout(pad=2.0)

#
# Plot streams here
#
plot_streams(ax1, V1, [-2,2,-2,2], 12)
ax1.set_title("Streams of V1")
plot_streams(ax2, V2, [-2,2,-2,2], 12)
ax2.set_title("Streams of V2")
plot_streams(ax3, V3, [-2,2,-2,2], 12)
ax3.set_title("Streams of V3")
plot_streams(ax4, V4, [-2,2,-2,2], 12)
ax4.set_title("Streams of V4")
```

Figure 2



Out[10]: Text(0.5, 1.0, 'Streams of V4')

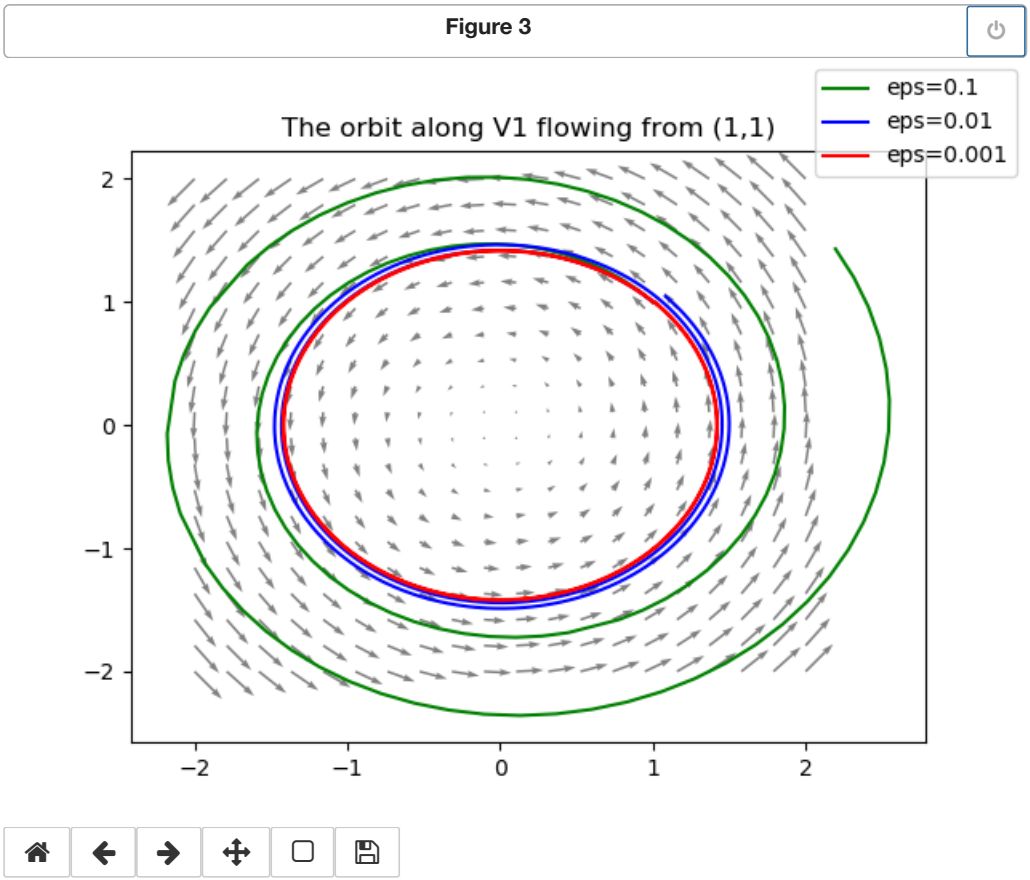
In [11]:
```
def euler(v_x, v_y, steps=10, eps=0.1):
    """Apply Euler's method `steps` number of times with a step-size of `eps` for V1"""
    for i in range(steps):
        x, y = V1(v_x, v_y)
        v_x = v_x + eps*x
        v_y = v_y + eps*y
    return v_x, v_y #approximate result of flowing along F for n*eps seconds

print("Starting at (1,0) with 1 steps, Euler's method gives", euler(1, 0, 1),
      "\nWith two steps", euler(1, 0, 2),
      "\nWith three steps", euler(1, 0, 3))
```

```
Starting at (1,0) with 1 steps, Euler's method gives (1.0, 0.1)
With two steps (0.99, 0.2)
With three steps (0.97, 0.29900000000000004)
```

In [12]:
```
def flow(v_x, v_y, F, t, eps=1/100):
    steps = int(t/eps)
    for i in range(steps):
        x, y = F(v_x, v_y)
        v_x = v_x + eps*x
        v_y = v_y + eps*y
    return v_x, v_y #approximate result of flowing along F for t seconds

def orbit(v_x, v_y, F, t_max=1, eps=1/100):
    vx_list = []
    vy_list = []
    steps = int(t_max/eps)
    for i in range(steps):
        new_xs, new_ys = flow(v_x, v_y, F, i*eps, eps)
        vx_list.append(new_xs)
        vy_list.append(new_ys)
    return np.array(vx_list), np.array(vy_list)

print("Starting at (1,0) and flowing for 3.14 second, Euler's method gives", flow(1, 0, V1, 3.14)
      "and a 4 step approximate orbit is", orbit(1, 0, V1, 3.14, eps=3.14/4))
```

```
Starting at (1,0) and flowing for 3.14 second, Euler's method gives (-1.0158216319242126, 0.001
724169909617117) and a 4 step approximate orbit is (array([ 1.      ,  1.      ,  0.383775, -0.
848675]), array([0.        , 0.785     , 1.57      , 1.87126338]))
```

In [13]:
```
#
# Plot V1
#
extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()
plot_vectorfield(ax, V1, extents=extents)

xs, ys = orbit(1, 1, V1, t_max=2*6.28, eps=0.1)
ax.plot(xs, ys, color="green", label="eps=0.1")
xs, ys = orbit(1, 1, V1, t_max=2*6.28, eps=0.01)
ax.plot(xs, ys, color="blue", label="eps=0.01")
xs, ys = orbit(1, 1, V1, t_max=2*6.28, eps=0.001)
ax.plot(xs, ys, color="red", label="eps=0.001")

ax.set_title("The orbit along V1 flowing from (1,1)")

fig.legend()
```



Out[13]: <matplotlib.legend.Legend at 0x7fedcd203048>

The simulated orbits are all periodic, while larger steps $\epsilon = 0.1$ (large epsilon) spirals out more since they are simulations, and infinitely small epsilons produce more accruate approximations.

In [14]:
```
def split_runs(seq):
    """Input a sequence `seq` and return a list of maximal runs contained in `seq`
    where a run is defined as a sequence that increases by exactly 1 each index."""
    result = []
    i = 1
    while i < len(seq):
        list = [seq[i-1]]
        while i < len(seq) and seq[i]-1 == seq[i-1]:
            list.append(seq[i])
            i+=1
        result.append(list)
        if i == len(seq)-1:
            result.append([seq[i]])
        i+=1
    return result

seq = [1,2,3,6,7,9,15,16,17,18,20]
print("Splitting the sequence", seq, "into runs gives is expected to give\n"
      " [[1, 2, 3], [6, 7], [9], [15, 16, 17, 18], [20]]\n"
      "and actually gives\n", split_runs(seq))
seq2= [1, 3, 6, 7, 8] # another test
print("Splitting the sequence", seq2, "into runs gives is expected to give\n"
      " [[1], [3], [6, 7, 8]]\n"
      "and actually gives\n", split_runs(seq2))
```

```
Splitting the sequence [1, 2, 3, 6, 7, 9, 15, 16, 17, 18, 20] into runs gives is expected to gi
ve
 [[1, 2, 3], [6, 7], [9], [15, 16, 17, 18], [20]]
and actually gives
 [[1, 2, 3], [6, 7], [9], [15, 16, 17, 18], [20]]
Splitting the sequence [1, 3, 6, 7, 8] into runs gives is expected to give
 [[1], [3], [6, 7, 8]]
and actually gives
 [[1], [3], [6, 7, 8]]
```

In [15]:
```
def indices_close_to_min(seq, tol=0.01):
    """returns the indices of values in `seq` that are within `tol`
    of the minimum value in `seq`"""
    result = []
    min_value = np.amin(seq)
    for i in range(len(seq)):
        if abs(seq[i] - min_value) <= tol:
            result.append(i)
    return result

seq = np.array([1, .8, .7, .701, .702, .73, .77])
print("The sequence", seq, "is close to its minimum at the indices", indices_close_to_min(seq))
```

```
The sequence [1.    0.8   0.7   0.701 0.702 0.73  0.77 ] is close to its minimum at the indices
[2, 3, 4]
```

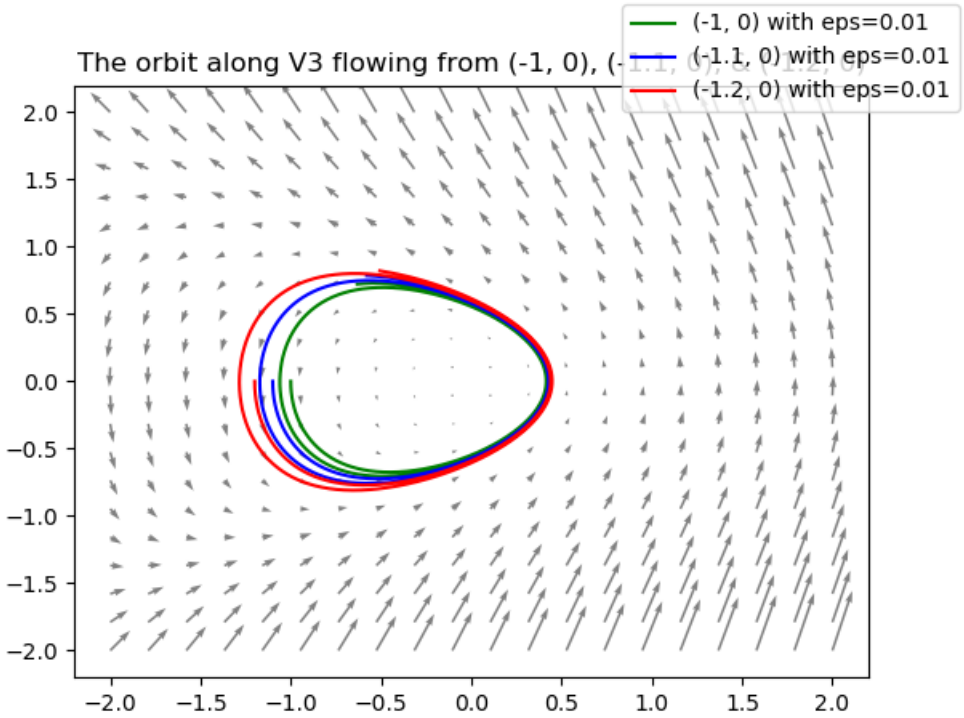
In [16]:
```
def guess_minimums(seq, eps=0.01, tol=0.01):
    """Given a list `seq` where each value is a step along the dynamical
    system with increment `eps`, estimate where the sequence attains a minimum"""
    indices = indices_close_to_min(seq, tol)
    run_list = split_runs(indices)

    result = []
    for run in run_list:
        for i in range(len(indices)):
            if indices[i] in run and len(run) > 1:
                poss_min = run[1]*eps
                if poss_min not in result:
                    result.append(poss_min)
            else:
                result.append(indices[i]*eps)
    return result

#
# Estimate the period of a (-1,0) flowing along V1
#
t = 3*np.pi #3pi, enough time for it to cross the period 2pi
eps1 = 0.01
xs_1, ys_1 = orbit(-1, 0, V1, t, eps1)
guess1_x = guess_minimums(xs_1, eps1)

eps2 = 0.001
xs_2, ys_2 = orbit(-1, 0, V1, t, eps2)
guess2_x = guess_minimums(xs_2, eps2, 0.001)


print("The period approximated by eps = 0.01 is\n", guess1_x)
print("The period approximated by eps = 0.001 is\n", guess2_x)
```

```
The period approximated by eps = 0.01 is
 [6.16]
The period approximated by eps = 0.001 is
 [6.2410000000000005]
```

The approximation of period (2π) is decent, with an increase in precision as the steps become smaller.

In [17]:
```
#
# Plot V3
#

extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()
plot_vectorfield(ax, V3, extents=extents)

xs, ys = orbit(-1, 0, V3, t_max=2*6.28, eps=0.01)
ax.plot(xs, ys, color="green", label="(-1, 0) with eps=0.01")
xs, ys = orbit(-1.1, 0, V3, t_max=2*6.28, eps=0.01)
ax.plot(xs, ys, color="blue", label="(-1.1, 0) with eps=0.01")
xs, ys = orbit(-1.2, 0, V3, t_max=2*6.28, eps=0.01)
ax.plot(xs, ys, color="red", label="(-1.2, 0) with eps=0.01")

ax.set_title("The orbit along V3 flowing from (-1, 0), (-1.1, 0), & (-1.2, 0)")

fig.legend()
```

Figure 4



Out[17]: <matplotlib.legend.Legend at 0x7fedcd18da20>

In [18]:
```
#
# Estimate the period of the flow along V3 starting at (-1,0), (-1.1, 0), and (-1.2,0)
#

#
# Estimate the period of a (-1,0) flowing along V3
#
t = 10
eps = 0.001
xs_1, ys_1 = orbit(-1, 0, V3, t, eps)
guess1_x = guess_minimums(xs_1, eps, eps)

print("The period of the flow along V3 starting at (-1, 0) approximated by eps = 0.001 is\n", gue

#
# Estimate the period of a (-1.1,0) flowing along V3
#

xs_2, ys_2 = orbit(-1.1, 0, V3, t, eps)
guess2_x = guess_minimums(xs_2, eps, eps)

print("The period of the flow along V3 starting at (-1.1, 0) approximated by eps = 0.001 is\n", g


#
# Estimate the period of a (-1.2,0) flowing along V3
#

xs_3, ys_3 = orbit(-1.2, 0, V3, t, eps)
guess3_x = guess_minimums(xs_3, eps, eps)

print("The period of the flow along V3 starting at (-1.2, 0) approximated by eps = 0.001 is\n", g
```

```
The period of the flow along V3 starting at (-1, 0) approximated by eps = 0.001 is
 [6.721]
The period of the flow along V3 starting at (-1.1, 0) approximated by eps = 0.001 is
 [6.797]
The period of the flow along V3 starting at (-1.2, 0) approximated by eps = 0.001 is
 [6.8740000000000006]
```

The flows starting near $(-1,0)$ (like $(-1.1,0),(-1.2,0)$) have similar period, however, is unstable, as the higer order terms y^2 dominate over time as the points flow along the vector field V3. Only the orgin is stable.

Flowing Areas

In [19]:
```
#
# Helper functions
#
def zip_coords(coords, arg2=None):
    """Give lists [(x1,x2,...), (y1,y2,...)] turn it into [(x1,y1), (x2,y2), ...]"""
    if arg2 is None:
        return list(zip(*coords))
    # if arg2 was specified, we called `zip_coords(xs, ys)` instead of `zip_coords( (xs, ys) )`
    return list(zip(coords, arg2))
```

In [20]:
```
def make_permiter_circle(center, rad, N=10):
    """Returns points along the perimiter of a circle centered at `center` with radus `rad`"""
    c1, c2 = center
    xs = []
    xy = []
    for i in range(N):
        x, y = rad*np.cos(2*np.pi/N*i)+c1, rad*np.sin(2*np.pi/N*i)+c2
        xs.append(x)
        xy.append(y)
    return xs, xy

print("The circle centered at (1,1) with radius (1/2) has points along the perimiter:",
      make_permiter_circle((1,1), 1/2, 4))
```

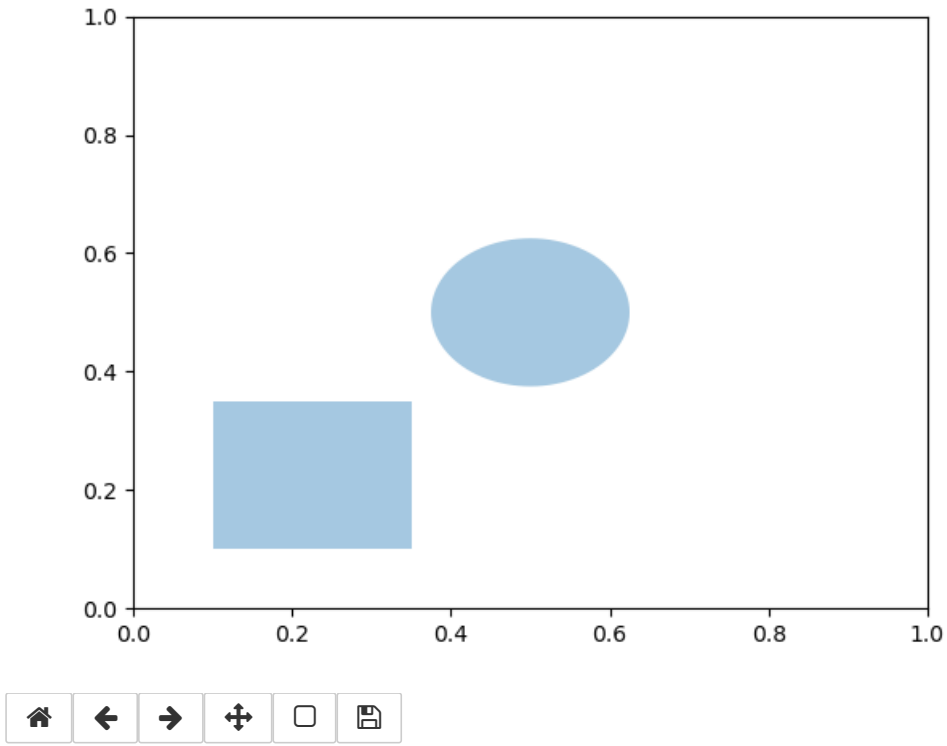
```
The circle centered at (1,1) with radius (1/2) has points along the perimiter: ([1.5, 1.0, 0.5,
0.9999999999999999], [1.0, 1.5, 1.0, 0.5])
```

In [21]:
```
def make_permiter_square(ll_corner, width, N=12):
    """Returns points along the perimiter of a square with lower-left corner `ll_corner`
    and widht `width`"""
    lx, ly = ll_corner
    xs = []
    xy = []
    num_per_side = int(N/4)
    side_dis = width / num_per_side

    #bottom side
    for i in range(num_per_side+1):
        x, y = lx + i*side_dis, ly
        xs.append(x)
        xy.append(y)
    #right side
    for i in range(num_per_side+1):
        x, y = lx + width, ly + i*side_dis
        xs.append(x)
        xy.append(y)
    #top side
    for _ in range(num_per_side+1):
        x, y = lx + width - i*side_dis, ly + width
        xs.append(x)
        xy.append(y)
    #left side
    for _ in range(num_per_side+1):
        x, y = lx, ly + width - i*side_dis
        xs.append(x)
        xy.append(y)
    return xs, xy
#     return np.linspace(0, 2*np.pi, N), np.linspace(0, 2*np.pi, N)

print("The circle centerd at (1,1) with radius (1/2) has points along the perimiter:",
      make_permiter_square((1,1), 1/2, 12))
```

```
The circle centerd at (1,1) with radius (1/2) has points along the perimiter: ([1.0, 1.16666666
66666667, 1.3333333333333333, 1.5, 1.5, 1.5, 1.5, 1.5, 1.0, 1.0, 1.0, 1.0, 1, 1, 1, 1], [1, 1,
1, 1, 1.0, 1.1666666666666667, 1.3333333333333333, 1.5, 1.5, 1.5, 1.5, 1.5, 1.0, 1.0, 1.0, 1.
0])
```

In [22]:
```
#
# Graph a circle and a square
#

from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection

fig, ax = plt.subplots()

# Matplotlib graphs "patches" consisting of polygons. Make a patch for both the circle and the sq
# and graph them.
circle = Polygon(zip_coords(make_permiter_circle((.5,.5), .125, 100)))
square = Polygon(zip_coords(make_permiter_square((0.1,0.1), .25, 100)))

patches = PatchCollection([circle, square], alpha=0.4)
ax.add_collection(patches)
```

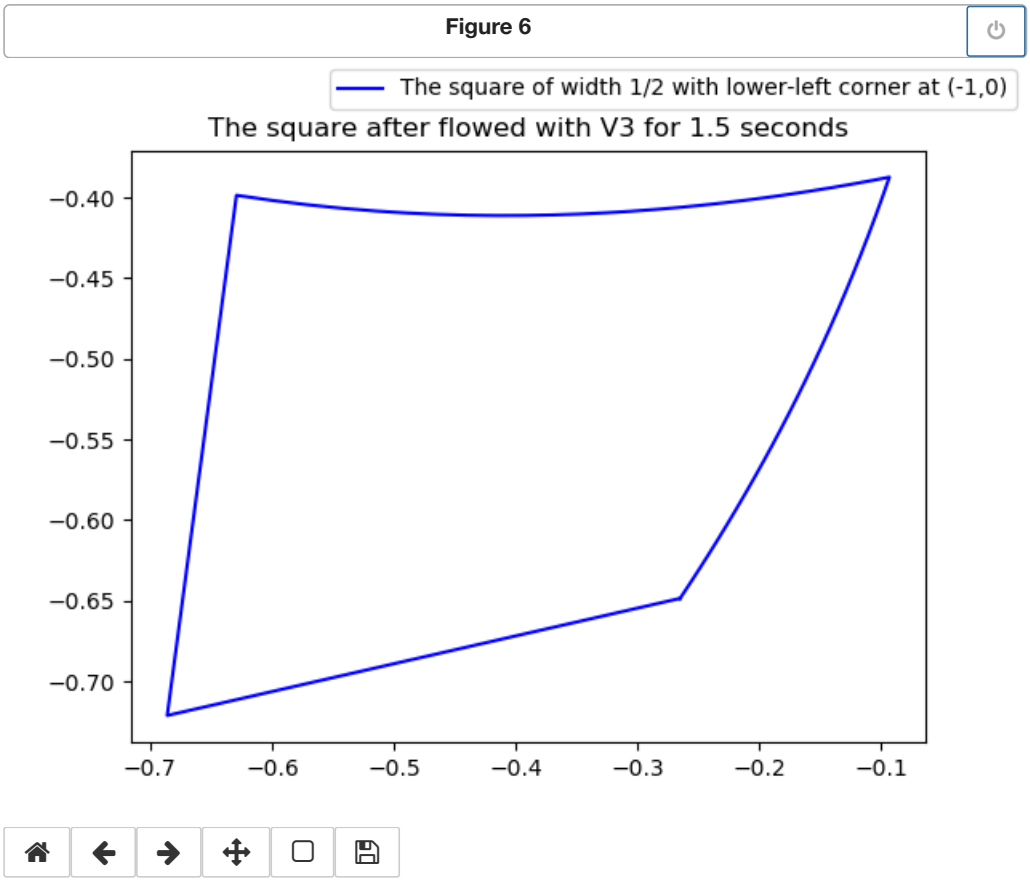
Figure 5



Out[22]: <matplotlib.collections.PatchCollection at 0x7fedcd1106a0>

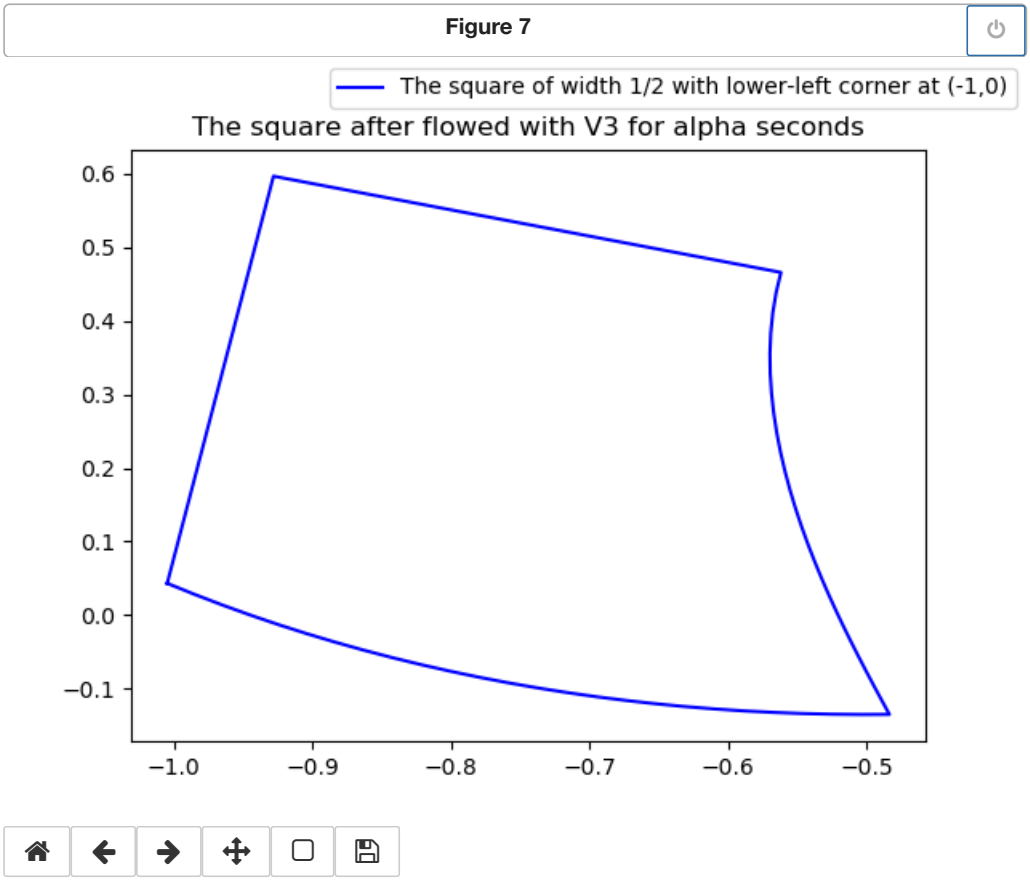
In [23]:
```
#
# Create a vectorized version of the `flow` function so it can work on lists of coordinates
#
vec_flow = np.vectorize(flow)
```

In [24]:
```
#
# Plot V3, the square, and the flow of a square for 1.5 seconds
#
xs, ys = make_permiter_square((-1,0), .50, 100)
extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()

xs, ys = vec_flow(xs, ys, V3, 1.5, eps=0.001)
ax.plot(xs, ys, color="blue", label="The square of width 1/2 with lower-left corner at (-1,0)")
ax.set_title("The square after flowed with V3 for 1.5 seconds")

fig.legend()
```



Out[24]: <matplotlib.legend.Legend at 0x7fedcd12dc18>

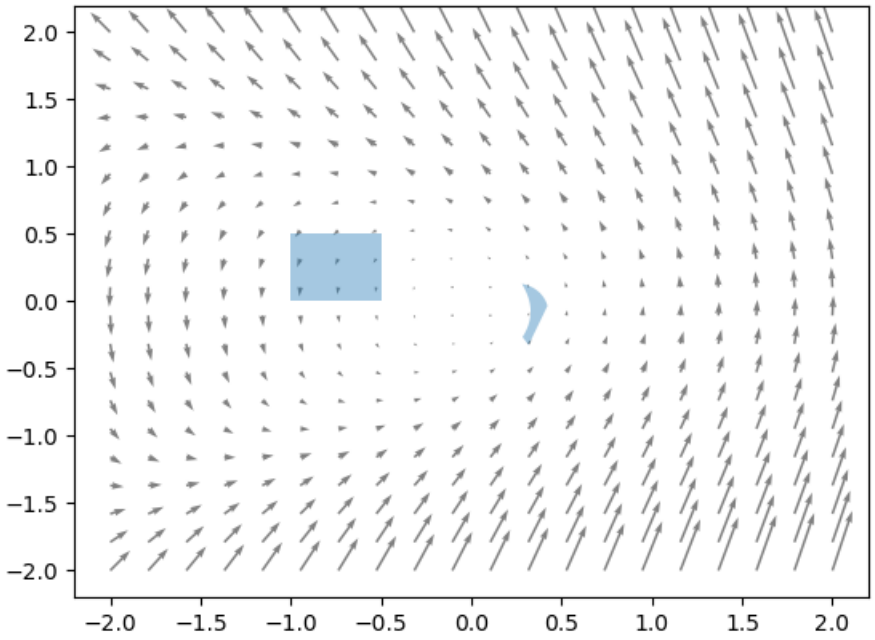
In [25]:
```
#
# Plot V3, the square, and the flow of a square for alpha seconds
#
alpha = 6.721 #from 6b

xs, ys = make_permiter_square((-1,0), .50, 100)
extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()

xs, ys = vec_flow(xs, ys, V3, alpha, eps=0.001)
ax.plot(xs, ys, color="blue", label="The square of width 1/2 with lower-left corner at (-1,0)")
ax.set_title("The square after flowed with V3 for alpha seconds")

fig.legend()
```



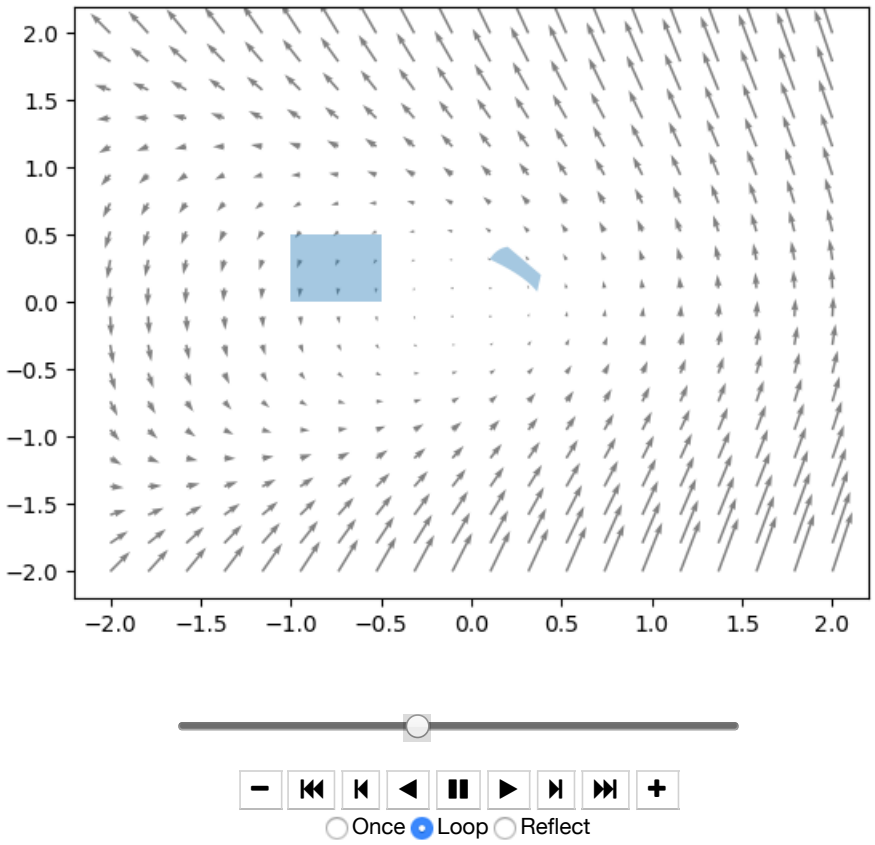
Out[25]: <matplotlib.legend.Legend at 0x7fedcd0e29e8>

In [26]:
```
#
# Animate the flow!
#

# these imports allow us to do animations
from matplotlib import animation, rc
from IPython.display import HTML

# duration of the flow and the framerate
FLOW_TIME = 10
FPS = 10
FLOW_PER_FRAME = 1/FPS

extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()
plot_vectorfield(ax, V3, extents=extents)

xs, ys = make_permiter_square((-1,0), .5, 100)
preflow = Polygon(zip_coords((xs, ys)))
patches = PatchCollection([preflow], alpha=0.4)
ax.add_collection(patches)

# the flowed coordinates
fxs, fys = xs, ys
def animate(i):
    # the flowed coordinates are global variables. We need this declaration to use them
    global fxs, fys
    # flow the coordinates a little bit
    fxs, fys = vec_flow(fxs, fys, V3, FLOW_PER_FRAME, 0.01)
    # draw the patches
    postflow = Polygon(zip_coords((fxs, fys)))
    patches.set_paths([preflow, postflow])
    return (patches,)

anim = animation.FuncAnimation(fig, animate, frames=FLOW_TIME*FPS, blit=True, interval=1000/FPS)
HTML(anim.to_jshtml())
```

Figure 8

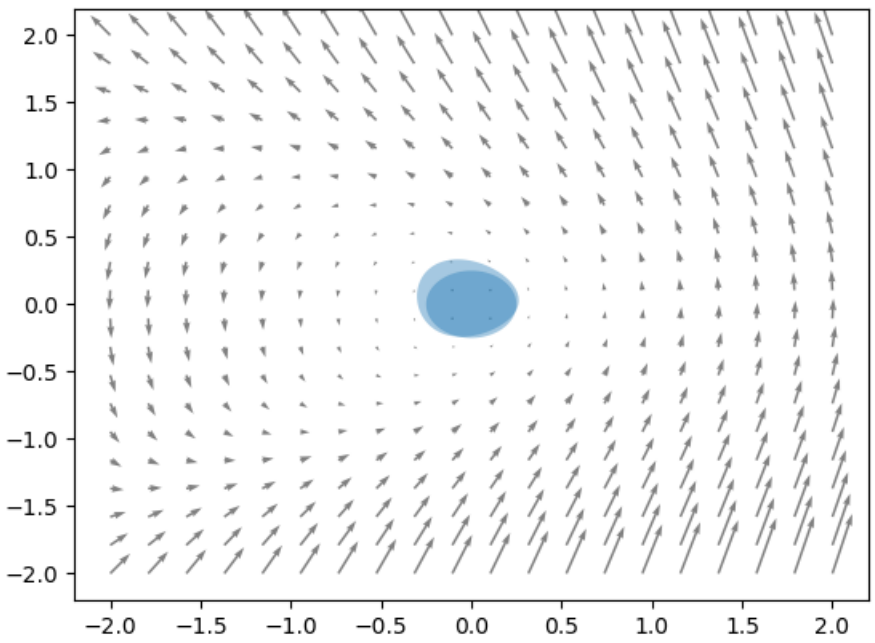


Out[26]:


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
Once Loop Reflect

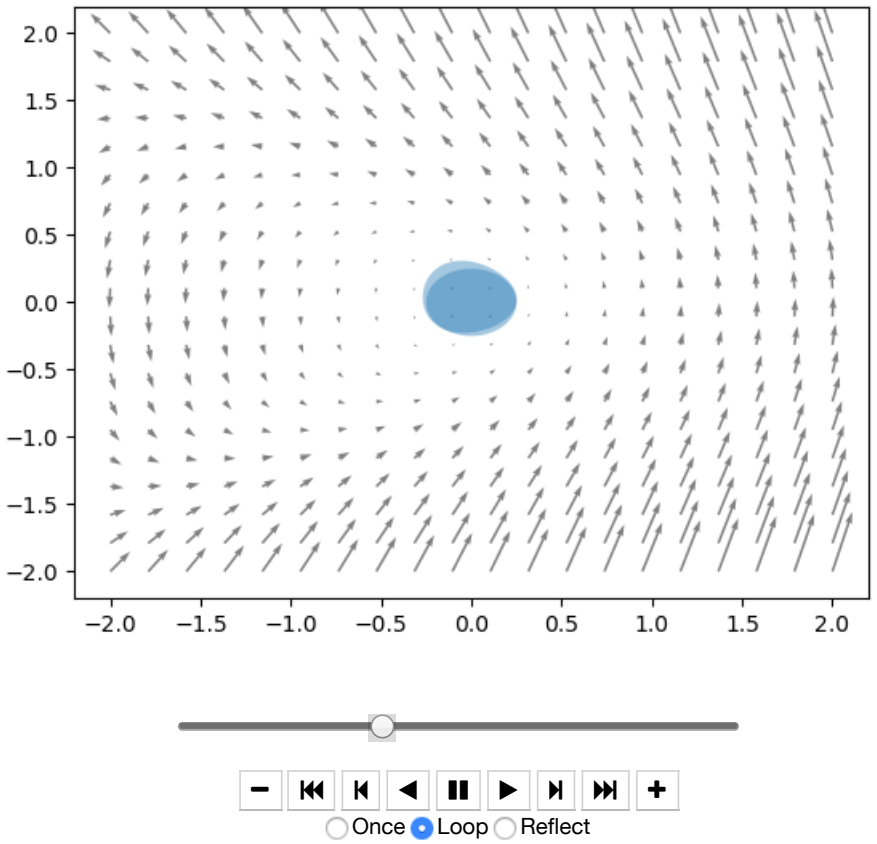
In [27]:
```
#
# Animate the flow of a circle of radius 1/4 at the origin
#

# duration of the flow and the framerate
FLOW_TIME = 20
FPS = 10
FLOW_PER_FRAME = 1/FPS

extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()
plot_vectorfield(ax, V3, extents=extents)

xs, ys = make_permiter_circle((0,0), .25, 100)
preflow = Polygon(zip_coords((xs, ys)))
patches = PatchCollection([preflow], alpha=0.4)
ax.add_collection(patches)

# the flowed coordinates
fxs, fys = xs, ys
def animate(i):
    # the flowed coordinates are global variables. We need this declaration to use them
    global fxs, fys
    # flow the coordinates a little bit
    fxs, fys = vec_flow(fxs, fys, V3, FLOW_PER_FRAME, 0.01)
    # draw the patches
    postflow = Polygon(zip_coords((fxs, fys)))
    patches.set_paths([preflow, postflow])
    return (patches,)

anim = animation.FuncAnimation(fig, animate, frames=FLOW_TIME*FPS, blit=True, interval=1000/FPS)
HTML(anim.to_jshtml())
```

Figure 9



```
Animation size has reached 21054291 bytes, exceeding the limit of 20971520.0. If you're sure yo
u want a larger animation embedded, set the animation.embed_limit rc parameter to a larger valu
e (in MB). This and further frames will be dropped.
```

Out[27]:



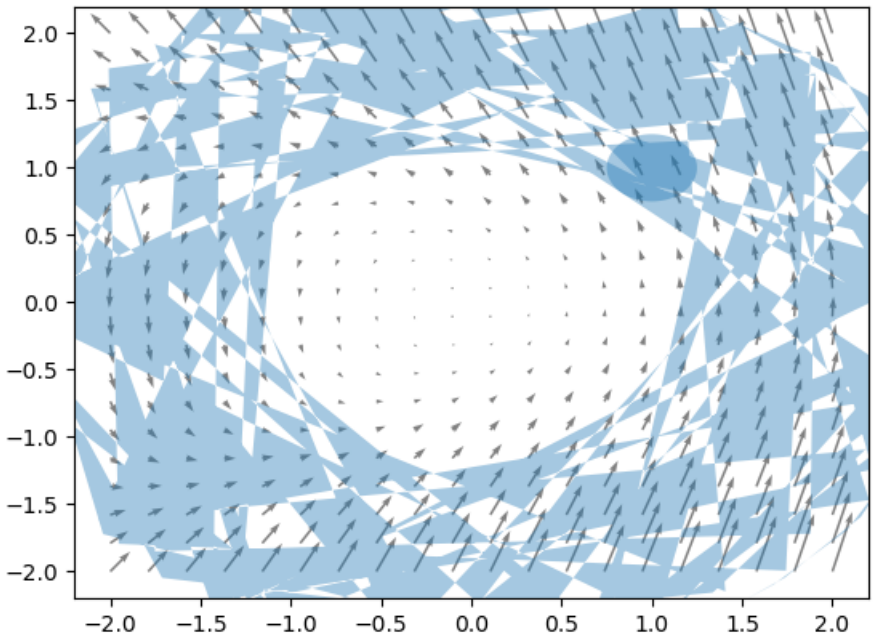
The origin is stable for this dynmical system that flow points along V3, as the circle goes back to the orginal circle while flowing for 20 seconds. This satisfies the $\epsilon - \delta$ definition of stable points.

The Rings of Saturn

In [28]:
```
#
# Create a vector field that simulates orbital velocity
#
def V5(X,Y):
    r = (X**2+Y**2)**(1/2)
    return 1/(r**(1/2))*1/r*(-Y), 1/(r**(1/2))*1/r*X
```

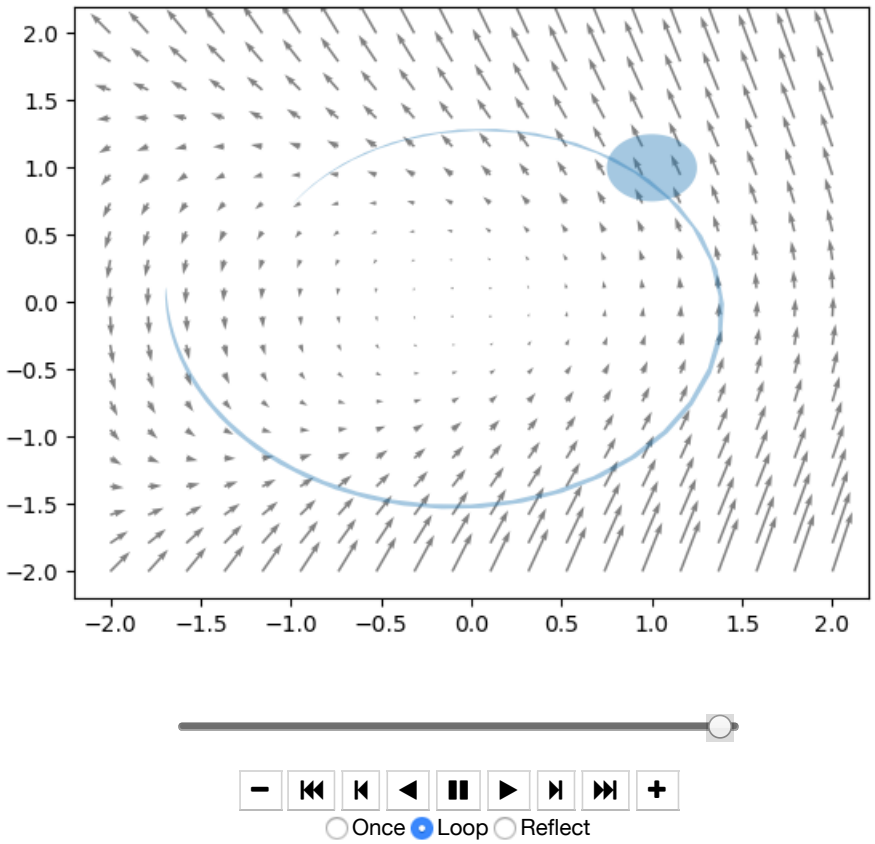
In [29]:
```
#
# Make an animation of an orbiting gas cloud
#

# duration of the flow and the framerate
FLOW_TIME = 20
FPS = 10
FLOW_PER_FRAME = 1/FPS

extents = [-2, 2, -2, 2]
fig, ax = plt.subplots()
plot_vectorfield(ax, V3, extents=extents)

xs, ys = make_permiter_circle((1,1), .25, 100)
preflow = Polygon(zip_coords((xs, ys)))
patches = PatchCollection([preflow], alpha=0.4)
ax.add_collection(patches)

# the flowed coordinates
fxs, fys = xs, ys
def animate(i):
    # the flowed coordinates are global variables. We need this declaration to use them
    global fxs, fys
    # flow the coordinates a little bit
    fxs, fys = vec_flow(fxs, fys, V5, FLOW_PER_FRAME, 0.01)
    # draw the patches
    postflow = Polygon(zip_coords((fxs, fys)))
    patches.set_paths([preflow, postflow])
    return (patches,)

anim = animation.FuncAnimation(fig, animate, frames=FLOW_TIME*FPS, blit=True, interval=1000/FPS)
HTML(anim.to_jshtml())
```

Figure 10




```
Animation size has reached 21077349 bytes, exceeding the limit of 20971520.0. If you're sure yo
u want a larger animation embedded, set the animation.embed_limit rc parameter to a larger valu
e (in MB). This and further frames will be dropped.
```

Out[29]:



By the above animation, we can clearly tell any circle/cluster/cloud of dust which started orbiting a planet becomes a ring over time. Same applies to Saturn who has many rings formed by dust near it.

In [ ]: