1. (a) $M = \begin{bmatrix} 0.1 & 0.2 & 0.4 \\ 0.6 & 0.6 & 0.5 \\ 0.3 & 0.2 & 0.1 \end{bmatrix}$

   (b) The eigenvalues are $\lambda_1 = 1, \lambda_2 = \frac{-1+\sqrt{2}}{10}, \lambda_3 = \frac{-1-\sqrt{2}}{10}$.

   The eigenvectors are $\text{span}(\vec{v}_{\lambda_1})$ where $\vec{v}_{\lambda_1} = \begin{bmatrix} 1.08333\ldots \\ 2.875 \\ 1 \end{bmatrix}$, $\text{span}(\vec{v}_{\lambda_2})$ where $\vec{v}_{\lambda_2} = \begin{bmatrix} 1.41421\ldots \\ -2.41421\ldots \\ 1 \end{bmatrix}$, $\text{span}(\vec{v}_{\lambda_3})$ where $\vec{v}_{\lambda_3} = \begin{bmatrix} -1.41421\ldots \\ 0.41421\ldots \\ 1 \end{bmatrix}$

   (c) Let's compute M^{3000} and see what $M^{3000}\vec{e_1}, M^{3000}\vec{e_2}, M^{3000}\vec{e_3}$ are to find the expected sale of relative proportions. We first diagonalize M, and get $M = PDP^{-1}$ where

$$P = \begin{bmatrix} 1.08333\ldots & 1.41421\ldots & -1.41421\ldots \\ 2.875 & -2.41421\ldots & 0.41421\ldots \\ 1 & 1 & 1 \end{bmatrix}, D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{-1+\sqrt{2}}{10} & 0 \\ 0 & 0 & -\frac{1+\sqrt{2}}{10} \end{bmatrix}$$

   and $P^{-1} = \begin{bmatrix} 0.20168\ldots & 0.20168\ldots & 0.20168\ldots \\ 0.17546\ldots & -0.17808\ldots & 032191\ldots \\ -0.37714\ldots & -0.02359\ldots & 0.47640\ldots \end{bmatrix}$

   Then,

$$M^{3000} = (PDP^{-1})^{3000} = PD^{3000}P^{-1}$$
$$= \begin{bmatrix} 0.2184859944 & 0.2184859944 & 0.218399328 \\ 0.57983 & 0.57983 & 0.5796 \\ 0.20168 & 0.20168 & 0.2016 \end{bmatrix}$$

   M^{3000} tells us that regardless of what customers wants initially, after watching 3000 times commercials, each customer would have a 22% chance to buy a fish product, 58% to buy a beef product, and 20% chance to buy a chicken product. Then, McDonald's can expect to sell 22% of the fish products, 58% of the beef products, and 20% of the chicken products.

   (d) Initial product preferences of a customer does not matter after the ads being play for at least 20 times (checked with a computer program) as $M^{20} = M^{3000}$. Similar to what is being mentioned in c), all the columns of M^{20} are the same, i.e., $M^{20}\vec{e_1} = M^{20}\vec{e_2} = M^{20}\vec{e_3}$, which means that regardless of what customers wants initially, after watching the commercial 20 times, each customer would have a 22% chance to buy a fish product, 58% to buy a beef product, and 20% chance to buy a chicken product. McDonald's should run the ad because the purchasing deal of fish, beef, and chicken has a $\frac{4}{18} = 22\%$, $\frac{10}{18} = 56\%$, and $\frac{4}{18} = 22\%$ proportion of the overall inventory respectively, which are very close to the expected selling amount (22% of the fish products, 58% of the beef products, and 20% of the chicken products) after running the ad to customers.

2. (a) Assume $\vec{q} = \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_n \end{bmatrix}$ is a probability vector, so $\sum_{i=1}^{n} q_i = 1 \quad \forall i \in \{1, 2, \dots, n\}$.

We know $P = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$ is an $n \times n$ stochastic matrix that has probability vectors as its columns with non-negative entries ($a_{ij} \geq 0$) that sum to one ($\sum_{i=1}^{n} a_{i1} = 1 \quad \forall i \in \{1, 2, \dots, n\}$).

Then $P\vec{q} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix}$ is a column vector that has each entry to be the dot product of the corresponding row with itself, i.e.

$$v_i = \begin{bmatrix} a_{i1} & a_{i2} & \dots & a_{in} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \dots \\ q_n \end{bmatrix} = a_{i1}q_1 + a_{i2}q_2 + \dots + a_{in}q_n \geq 0 \quad \forall i \in \{1, 2, \dots n\}$$

since a_{ij}, q_i's are non-negative, we know that the sum of their product is non-negative, so v_i's are non-negative. Also, $\sum_{i=1}^{n} v_i = 1$ because

$$\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} (a_{i1}q_1 + a_{i2}q_2 + \dots + a_{in}q_n) = \sum_{i=1}^{n} (q_1(a_{i1}) + q_2(a_{i2}) + \dots + q_n(a_{in}))$$

$$= q_1 \sum_{i=1}^{n} a_{i1} + q_2 \sum_{i=1}^{n} a_{i2} + \dots + q_n \sum_{i=1}^{n} a_{in} = q_1 + q_2 + \dots + q_n = 1$$

since we know $\sum_{i=1}^{n} a_{i1} = 1$ and $\sum_{i=1}^{n} q_i = 1 \quad \forall i \in \{1, 2, \dots, n\}$.

Thus, $\vec{q}$ is a probability vector.

(b) We want to prove that P^k is a stochastic matrix for all $k \geq 0$.

Proof by induction:

Base case: Suppose $k = 0$, then $P^k = p^0 = I$ which is clearly a stochastic matrix with non-negative entries and each column sums to 1.

Induction step: Assume for all $k \geq 0$, P^k is a stochastic matrix, we want to show that P^{k+1} is also a stochastic matrix.

$P^{k+1} = P^k P$ is a product of two stochastic matrices, P^k (by assumption and P is given in the question).

Suppose $P = \begin{bmatrix} | & | & & | \\ \vec{p_1} & \vec{p_2} & \dots & \vec{p_n} \\ | & | & & | \end{bmatrix}$, where $\vec{p_i}$ are probability vectors.

Then, $P^k P = \begin{bmatrix} | & | & & | \\ P^k\vec{p_1} & P^k\vec{p_2} & \dots & P^k\vec{p_n} \\ | & | & & | \end{bmatrix}$.

By a), we know that each resulting column of the desired matrix $P^k P$ is a probability vector.

Thus, P^{k+1} is a stochastic matrix.

This concludes our induction proof.

(c) Suppose P has a left eigenvector $\vec{w}$ with eigenvalue 1, i.e., $\vec{w}P = \vec{w}$. Then, $\vec{w}P = \vec{w}I$, which means $\vec{w}P - \vec{w}I = \vec{0} = \vec{w}(P - I)$. Since $\vec{w}$ is a non-zero row vector, we know

$P - I = 0_{n,n}$, the zero matrix, which means $P = I$. We can conclude that P has a left eigenvector with eigenvalue 1 if and only if P is the identity matrix (identity matrix is also a stochastic matrix).

(d) Suppose P has a left eigenvector (non-zero) $\vec{w}$ with eigenvalue λ_w such that $\vec{w}P = \lambda_w\vec{w}$, and a right eigenvector (non-zero) $\vec{v}$ with eigenvalue λ_v such that $P\vec{v} = \lambda_v\vec{v}$. Then, we know $\vec{w}(P - \lambda_w I) = \vec{0}$ and $(P - \lambda_v I)\vec{v} = \vec{0}$. We also know that finding the eigenvalues λ_w, λ_v is to compute the determinant of $P - \lambda_w I$ and $P - \lambda_v I$ which will result in the same characteristic polynomials, and the eigenvalues will be the same since they are the roots of the same characteristic equations.

(e) We want to show that $\mathcal{P}(S) \subseteq S$, where $\mathcal{P}(\vec{x}) = P\vec{x}$ and S is the unit n-symplex.
First, notice that the vectors in S are probability vectors because

$$\forall \vec{s} \in S, \vec{s} = \alpha_1\vec{e}_1 + \alpha_2\vec{e}_2 + \cdots + \alpha_n\vec{e}_n = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \cdots \\ \alpha_n \end{bmatrix}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0$ and $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$ by definition, which satisfies the definition of a probability vector that has non-negative entries which sum to 1. We know from a) that if $\vec{s} \in S$, then $\mathcal{P}(\vec{s}) = P\vec{s}$ is a probability vector since P is a stochastic matrix. We also know from $3b$) that S is the set of all probability vectors in $\mathbb{R}^n$, so $P\vec{s} \in S$, i.e., $\mathcal{P}(S) \subseteq S$, as needed.

(f) In order to use the Brouwer Fixed-point Theorem, we want to show that the linear transformation $\mathcal{P}|_S : S \to S$ is continuous, i.e.

$$\forall \epsilon > 0, \exists \delta > 0 \text{ s.t. } \|\vec{x} - \vec{y}\| < \delta \implies \|\mathcal{P}|_S(\vec{x}) - \mathcal{P}|_S(\vec{y})\| < \epsilon$$

Fix $\epsilon > 0$, we know $\mathcal{P}|_S$ is still a linear transformation, i.e.

$$\exists M > 0 \text{ s.t. } \|\mathcal{P}|_S(\vec{x})\| \leq M\|\vec{x}\| \quad \forall \vec{x} \in S$$

Pick $\delta = \frac{\epsilon}{M}$, assume $\|\vec{x} - \vec{y}\| < \delta$, then

$$\|\mathcal{P}|_S(\vec{x}) - \mathcal{P}|_S(\vec{y})\| = \|\mathcal{P}|_S(\vec{x} - \vec{y})\| \leq M\|\vec{x} - \vec{y}\| < M \cdot \delta = M \cdot \frac{\epsilon}{M} = \epsilon$$

by $\mathcal{P}|_S$ is a linear transformation and assumption.
Thus, $\mathcal{P}|_S : S \to S$ is continuous, and by Brouwer Fixed-point Theorem, we know it has at least one fixed point, i.e.,

$$\exists \vec{x} \in S \text{ s.t. } \mathcal{P}|_S(\vec{x}) = \vec{x}$$

We also know that

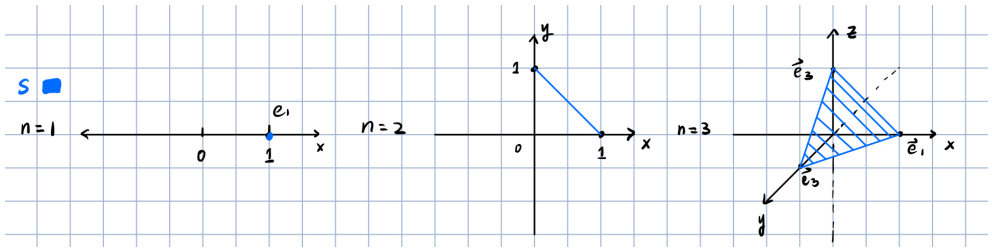
$$\mathcal{P}|_S(\vec{x}) = P\vec{x}$$

Then

$$\mathcal{P}|_S(\vec{x}) = P\vec{x} = \vec{x} = 1\vec{x}$$

This tells us that $\vec{x}$ is an eigenvector of P with eigenvalue 1 since $\vec{x}$ is a probability vector that is non-zero, as needed.

3. (a)



(b) Let A to be the set of all probability vectors of $\mathbb{R}^n$, we want to show that $S \subseteq A$ and $A \subseteq S$.

First, we want to show that $S \subseteq A$. Fix $\vec{x} \in S$. We want to show that $\vec{x} \in A$. We know, from $2e$), that the vectors in S are probability vectors because

$$\forall \vec{s} \in S, \vec{s} = \alpha_1\vec{e}_1 + \alpha_2\vec{e}_2 + \cdots + \alpha_n\vec{e}_n = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \cdots \\ \alpha_n \end{bmatrix}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0$ and $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$ by definition of being a convex linear combination of the standard basis vectors, which satisfies the definition of being a probability vector that has non-negative entries which sum to 1.

Next, we want to show that $A \subseteq S$. Fix $\vec{a} \in A$ where

$$\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix} = a_1 \begin{bmatrix} 1 \\ 0 \\ \cdots \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ \cdots \\ 0 \end{bmatrix} + \cdots + a_n \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 1 \end{bmatrix} = a_1\vec{e}_1 + a_2\vec{e}_2 + \cdots + a_n\vec{e}_n$$

We know $\vec{a} \in A$ as a probability vector has non-negative entries, so $a_1, a_2, \ldots a_n \geq 0$ and sum to 1, so $a_1 + a_2 + \cdots + a_n = 1$, which by definition, is a convex linear combination of the standard basis vectors, $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n$. Thus, $\vec{a} \in S$.

We can finally conclude that $A = S$, as needed.

(c) Assume $\vec{p} = \begin{bmatrix} p_1 \\ p_2 \\ \cdots \\ p_n \end{bmatrix}, \vec{q} = \begin{bmatrix} q_1 \\ q_2 \\ \cdots \\ q_n \end{bmatrix} \in S$.

We know $p_1, p_2, \ldots, p_n, q_1, q_2, \ldots q_n \geq 0$ and $p_1 + p_2 + \cdots + p_n = 1, q_1 + q_2 + \cdots + q_n = 1$ by $3b$) [vectors in S are probability vectors in $\mathbb{R}^n$].

We want to show that all convex linear combinations of $\vec{p}, \vec{q}$ are in S.

Fix a convex linear combinations $\vec{v}$ of $\vec{p}, \vec{q}$, i.e.,

$$\vec{v} = \beta_1\vec{p} + \beta_2\vec{q}, \quad \text{where } \beta_1, \beta_2 \geq 0, \beta_1 + \beta_2 = 1$$

We want to show that $\vec{v} \in S$, i.e., it can be written as a convex linear combinations of the standard basis vectors, or it is a probability vector by $3b$).

We know

$$\vec{v} = \beta_1\vec{p} + \beta_2\vec{q} = \beta_1 \begin{bmatrix} p_1 \\ p_2 \\ \cdots \\ p_n \end{bmatrix} + \beta_2 \begin{bmatrix} q_1 \\ q_2 \\ \cdots \\ q_n \end{bmatrix} = \begin{bmatrix} \beta_1 p_1 + \beta_2 q_1 \\ \beta_1 p_2 + \beta_2 q_2 \\ \cdots \\ \beta_1 p_n + \beta_2 q_n \end{bmatrix}$$

We know the entries of $\vec{v}$ are non-negative since $\beta_1, \beta_2, p_1, p_2, \ldots, p_n, q_1, q_2, \ldots q_n \geq 0$, and sum to 1 because

$$\beta_1 p_1 + \beta_2 q_1 + \beta_1 p_2 + \beta_2 q_2 + \cdots + \beta_1 p_n + \beta_2 q_n$$

$$= \beta_1 (p_1 + p_2 + \cdots + p_n) + \beta_2 (q_1 + q_2 + \cdots + q_n) = \beta_1 + \beta_2 = 1$$

by $p_1 + p_2 + \cdots + p_n = 1, q_1 + q_2 + \cdots + q_n = 1$ and $\beta_1 + \beta_2 = 1$.

So $\vec{v}$ is a probability vector, i.e., $\vec{v} \in S$, as needed.

(d) Assume $n \geq 2$. Let $\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix}, \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix} \in S$ be distinct points and let $\ell \subseteq \mathbb{R}^n$ be the line passing through $\vec{a}$ and $\vec{b}$.

We want to show that ℓ intersects the boundary of S.

First, let's parameterize the line ℓ in vector form: $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_i \\ \cdots \\ x_n \end{bmatrix} = t(\vec{b} - \vec{a}) + \vec{a} = t(\vec{b} - \vec{a}) + \vec{a}$

where $t \in \mathbb{R}$.

The direction vector $\vec{b} - \vec{a}$ is non-zero since $\vec{a}, \vec{b} \in S$ are non-zero distinct points.

Next, we want to show $\exists t \in \mathbb{R}$ s.t. $\vec{x} = t(\vec{b} - \vec{a}) + \vec{a} \in \partial S$. We know ∂S consists of all vectors in S where at least one coordinate is zero. Thus, we need to show that $\vec{x}$ is a probability vector (again from $3b$)) where at least one coordinate is 0.

Notice that the sum of all entries of any $\vec{x}$ is 1:

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} (t(b_i - a_i) + a_i) = \sum_{i=1}^{n} (tb_i - ta_i + a_i)$$

$$= t(b_1 + b_2 + \cdots + b_n) - t(a_1 + a_2 + \cdots + a_n) + (a_1 + a_2 + \cdots + a_n)$$

$$= t - t + 1 = 1$$

Consider $v_i(t) = t(b_i - a_i) + a_i = x_i \quad \forall i \in \{1, 2, \ldots, n\}$, it is clear to see that each $v_i(t)$ is continues. We also know the $\min_{i \in \{1,2,\ldots,n\}} \{v_i(t)\}$ is also a continues function by the fact that each $v_i(t)$ is continues.

Then, we know $\exists t$ such that one of the $v_i(t)$ will be 0 while the other entries are non-negative by Intermediate Value Theorem. This is because as the line ℓ going out of S, one of the entry of $\vec{x}$ has to become negative.

Then, we know ℓ intersects the boundary of S at $\vec{x}$ for such t, as needed.

(e) Assume $V \subseteq \mathbb{R}^n$ is a subspace of dimension at least two. Assume $V \cap S$ is nonempty, we want to show that $V \cap \partial S$ is non-empty.

Case 1: It is clear to show that $V \cap \partial S$ is non-empty if there are two distinct vectors in $V \cap S$. We know that the line ℓ that passes through these two vectors is also in V by V being a subspace (closed under scalar multiplication and vector addition), and by $3d$), ℓ intersects ∂S.

Case 2: Assume there is only one vector in $V \cap S$, i.e. $\vec{x} \in V \cap S$. We want to show that there is another distinct vector in $V \cap S$ so that we can go back to case 1.

Claim: There exists a non-zero vector $\vec{y} \in V$ such that $\vec{x} + \vec{y} \in S$.

We know the sum of the entries of $\vec{x} + \vec{y}$ is 1 by definition of being in S, and sum of the entries $\vec{x}$ is also 1. Then, the entries of $\vec{y}$ sum to 0.

Consider $Y = \{\vec{y} \in \mathbb{R}^n : \text{entries of } \vec{y} \text{ sum to } 0\}$. Notice that Y is a subspace of $\mathbb{R}^n$ as it is closed under vectors addition and scalar multiplication, and $Y \cap V$, $Y + V$ are both subspaces of $\mathbb{R}^n$ by Y, V being subspaces, so talking about their dimensions is valid.

We want to show that $\dim(Y \cap V) \geq 1$ so that there exists a non-zero vector $\vec{y} \in Y \cap V$, and to conclude that $\vec{x} + \vec{y} \in S$.

We know $\dim(Y) = n-1$ because

$$Y = \{\vec{y} \in \mathbb{R}^n : \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_{n-1} \\ -(y_1 + y_2 + \dots + y_{n-1}) \end{bmatrix} \text{ for some } y_1, y_2, \dots, y_{n-1} \in \mathbb{R}\}$$

and a basis γ for $Y \subset \mathbb{R}^n$ is

$$\gamma = \{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \dots \\ 0 \\ -1 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 1 \\ -1 \end{bmatrix} \}, |\gamma| = n-1$$

since

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_{n-1} \\ -(y_1 + y_2 + \dots + y_{n-1}) \end{bmatrix} = y_1 \begin{bmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{bmatrix} + y_2 \begin{bmatrix} 0 \\ 1 \\ 0 \\ \dots \\ 0 \\ -1 \end{bmatrix} + \dots + y_{n-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 1 \\ -1 \end{bmatrix}$$

We also know $Y + V$ is a subspace of $\mathbb{R}^n$, so

$$\begin{aligned} &\dim(Y+V) = \dim(Y) + \dim(V) - \dim(Y \cap V) \\ \iff &\dim(Y \cap V) = \dim(Y) + \dim(V) - \dim(Y+V) \\ &\geq (n-1) + 2 - \dim(Y+V) \\ &= n + 1 - \dim(Y+V) \end{aligned}$$

since $\dim(V) \geq 2$ by assumption and $\dim(Y) = n-1$. Both Y, V are subspaces such that $Y, V \subset \mathbb{R}^n$, we know $\dim(Y+V) \leq \dim(\mathbb{R}^n) = n \iff -\dim(Y+V) \geq -n$. Then

$$\dim(Y \cap V) \geq n + 1 - \dim(Y+V) = n + 1 - n = 1$$

So we know $\dim(Y \cap V) \geq 1$, as needed.

By claim, we know that there exists a non-zero vector $\vec{y} \in V$ ($\vec{y} \in Y \cap V$ whose entries sums to 0) such that $\vec{x} + \vec{y} \in S$, then $\vec{x} + \vec{y} \in V$ as V is a subspace that is closed under vectors addition. So $\vec{x} + \vec{y} \in V \cap S$ is a distinct vector than $\vec{x}$, which brings us back to case 1.

(f) Assume $\vec{a}, \vec{b} \in S$ are distinct eigenvectors for P with eigenvalue 1. We want to show that there exists a vector $\vec{d} \in \partial S$ which is also an eigenvector for P with eigenvalue 1. We know eigenvectors for P with eigenvalue 1 are probability vectors from $2f$). Then from $3d$), we know the line ℓ passing through $\vec{a}, \vec{b}$ intersects ∂S at at least one point, say $\vec{x}$.

We know $\vec{x} = t(\vec{b} - \vec{a}) + \vec{a}$ for some $t \in \mathbb{R}$ as it is in ℓ. Then

$$P\vec{x} = P(t(\vec{b} - \vec{a}) + \vec{a}) = t(P(\vec{b}) - P(\vec{a})) + P(\vec{a}) = t(\vec{b} - \vec{a}) + \vec{a} = \vec{x}$$

by linearity of $\mathcal{P}$ whose matrix representation is P and $\vec{a}, \vec{b}$ are eigenvectors of P with eigenvalue 1.

Thus, $\vec{x}$ is an eigenvector for P with eigenvalue 1 and it is in ∂S. Notice that $\vec{x}$ is non-zero because $\vec{0} \notin \partial S$.

4. Let $\mathcal{M} = (M_0, M_1, \ldots)$ be a stationary Markov chain on a graph $\mathcal{G}$ with n vertices, and let P be the (stochastic) transition matrix for $\mathcal{M}$. Further, suppose $\mathcal{M}$ is modeled by the dynamical system (T, Ω), where Ω is the space of probability distributions on the n vertices.

   (a) Example 1: G is a graph with no edges between 2 vertices, only self-loops, where its corresponding stochastic transition matrix is

   $$P = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

   It is clear to see that $\lim_{k\to\infty} P^k$ exists and is equal to P.
   Example 2: G is a graph with only edges going to the other vertex so that its corresponding stochastic transition matrix is

   $$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

   It is clear to see that $\lim_{k\to\infty} P^k$ does not exist as it alternates between I (for k is even) and P (for k is odd).
   The conditions on $\mathcal{M}$ so $\lim_{k\to\infty} P^k$ exists is if its corresponding P has eigenvalues λ, then $|\lambda| < 1$ or $\lambda = 1$.
   If this condition fails as there is a $\lambda \leq -1$ or $\lambda > 1$, we know $\lim_{k\to\infty} \lambda^k$ does not exist (alternates if $\lambda \leq -1$, goes to infinity if $\lambda > 1$).
   Let $\vec{v}$ be an eigenvector of P with such λ. Suppose $\lim_{k\to\infty} P^k = P'$ exists, then $P'\vec{v} = (\lim_{k\to\infty} P^k)\vec{v} = \lim_{k\to\infty}(P^k\vec{v}) = \lim_{k\to\infty}(\lambda^m\vec{v}) = (\lim_{k\to\infty}\lambda^m)\vec{v}$ diverges as $\lim_{k\to\infty}\lambda^k$ does not exist.

   (b) Example 1: M which has this stochastic transition matrix:

   $$P = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

   has exactly 1 stationary distribution, $\vec{e}_1$.
   It is impossible to have exactly 2 or 3 stationary distributions for $\mathcal{M}$ since by 4*c*), if we have at least two stationary distributions, any convex linear combination of them is also a stationary distribution of $\mathcal{M}$.

   (c) Fix a convex linear combination $\vec{v}$ of stationary distributions $\vec{v}_1, \ldots \vec{v}_n$ so that

   $$\vec{v} = \alpha_1\vec{v}_1 + \alpha_2\vec{v}_2 + \cdots + \alpha_n\vec{v}_n$$

   where $\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0$ and $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$
   We want to show that $\vec{v}$ is a stationary distribution for $\mathcal{M}$, i.e. $P\vec{v} = \vec{v}$ where P is the transition matrix for $\mathcal{M}$.
   Then, we know

   $$P\vec{v} = P(\alpha_1\vec{v}_1 + \alpha_2\vec{v}_2 + \cdots + \alpha_n\vec{v}_n) = \alpha_1 P\vec{v}_1 + \alpha_2 P\vec{v}_2 + \cdots + \alpha_n P\vec{v}_n$$

   $$= \alpha_1\vec{v}_1 + \alpha_2\vec{v}_2 + \cdots + \alpha_n\vec{v}_n = \vec{v}$$

   by linearity of P and $\vec{v}_i$'s are stationary distributions where $P\vec{v}_i = v_i$, as needed.

   (d) We want to show that $\mathcal{M}$ always has at least one stationary distribution, i.e., its T has at least one fixed point on Ω, which is equivalent as showing its P has at least one fixed point on S_n, where S_n is the set of all probability distributions in Ω. We know by Brouwer fixed point theorem on $P : S_n \to S_n$, P has at least one fixed point as it is continuous from S_n to S_n, as needed.

(e) Assume $\mathcal{M}$ is primitive, we want to show that every stationary distribution for $\mathcal{M}$ must have full support, i.e., none of the entries of the stationary distribution is 0.

Fix a stationary distribution $\vec{x}$. We know by definition this means it is a fixed point of T, which is equivalent as $P\vec{x} = \vec{x}$. Suppose $\vec{x}$ does not have full support for the sake of contradiction, so i^{th} entry $x_i = 0$ for some $i \in \{1, 2, \ldots, n\}$.

By assumption, we know there exists a $k \in \mathbb{N}$ such that the probability of transitioning from state i to state j in exactly k steps is positive for every i and j, i.e., the matrix P^k has positive entries only. We also know that $P^k\vec{x} = \vec{x}$ by $\vec{x}$ being a stationary distribution. Then

$$P^k\vec{x} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{i1} & a_{i2} & \ldots & a_{in} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_i \\ \ldots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_i \\ \ldots \\ x_n \end{bmatrix} = \vec{x}$$

Since $a_{i1}, a_{i2}, \ldots, a_{in} > 0$, and $\vec{x}$ is also a probability vector whose entries sum to 1 such that at least one of the entries x_j is non-zero for some $j \neq i$, the i^{th} entry of $P^k\vec{x}$ is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ii}x_i + \cdots + a_{in}x_n = x_i > 0$. This contradicts with our assumption of $x_i = 0$.

(f) Assume $\mathcal{M}$ is primitive, we want to show that $\mathcal{M}$ has a unique stationary distribution.

From 4d), we know $\mathcal{M}$ has at least one stationary distribution, so its existence is guaranteed.

Next, we want to show its uniqueness.

Suppose not for the sake of contradiction. Then, we can find at least two distinct stationary distributions $\vec{a}, \vec{b} \in S$ for $\mathcal{M}$. Notice that they are also eigenvectors for P with eigenvalue 1 by being stationary distributions as $P\vec{a} = \vec{a}$ and $P\vec{b} = \vec{b}$.

Then by 3f), there exists a $\vec{d} \in \partial S$ which is also an eigenvector for P with eigenvalue 1, so we know $\vec{d}$ is a stationary distribution by definition of $\vec{d} \in \partial S$

From 4e), since $\mathcal{M}$ is primitive, we know all stationary distributions of $\mathcal{M}$ have full support, i.e. have non-zero entries. This includes $\vec{a}, \vec{b}, \vec{d}$. However, $\vec{d}$ does not have full support as it is in the boundary of S which consists of all vectors in S where at least one coordinate is zero.

This is a contradiction.

Then, we know M has a unique stationary distribution

(g) Assume $\mathcal{M}$ is primitive and $\lim_{k\to\infty} P^k = P'$ exists, We want to show that $P' = [\vec{s}|\vec{s}|\cdots|\vec{s}]$, where $\vec{s}$ is the unique stationary distribution for $\mathcal{M}$.

We know $PP' = P'$ since $PP' = P \lim_{k\to\infty} P^k = \lim_{k\to\infty} P^{k+1} = P'$. From 4$f$), we know $\mathcal{M}$ has a unique stationary distribution $\vec{s}$, which means the eigenspace for eigenvector of its P with eigenvalue 1 has dimension 1, as the normalized basis for the eigenspace of P with eigenvalue 1 contains $\vec{s}$ only.

Since $PP' = P'$, we know each column of P' is an eigenvector of P with eigenvalue 1, since if $P' = \begin{bmatrix} | & | & & | \\ P'_1 & P'_2 & \ldots & P'_n \\ | & | & & | \end{bmatrix}$, then

$$PP' = \begin{bmatrix} | & | & & | \\ PP'_1 & PP'_2 & \ldots & PP'_n \\ | & | & & | \end{bmatrix} = P' = \begin{bmatrix} | & | & & | \\ P'_1 & P'_2 & \ldots & p'_n \\ | & | & & | \end{bmatrix}$$

We get $PP'_i = P'_i \quad \forall i \in \{1, \ldots, n\}$. Since P'_i's are probability vectors (proven previously), they are also non-zero eigenvectors for P that has eigenvalue 1.

We know all eigenvectors that are also probability vectors for P with eigenvalue 1 has to be $\vec{s}$ by uniqueness, so $P'_i = \vec{s} \quad \forall i \in \{1, \ldots, n\}$, which means $P' = [\vec{s}|\vec{s}|\ldots|\vec{s}]$, as needed.

Homework 2

Do the programing part of Homework 2 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.

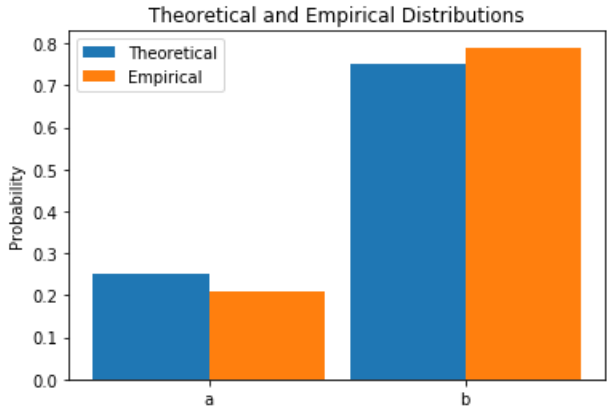
In [1]:
```
# import numpy and matplotlib
import numpy as np
import matplotlib.pylab as plt
from collections import Counter
# We give the matplotlib instruction twice, because firefox sometimes gets upset if we don't.
# note these `%`-commands are not actually Python commands. They are Jupyter-notebook-specific co
%matplotlib notebook
%matplotlib notebook
```

In [2]:
```
#
# Some helper functions
#

def plot_distributions(dists, labels=[], title=""):
    """Plot a bar graph of several distributions side-by-side. It is
    assume that the states in each distribution are in the same order.

    `dists` is a list of distributions
    `labels` is an optional list of labels, one for each distribution
    `title` is an optional title for the plot
    """
    states = [state for p,state in dists[0]]
    num_states = len(states)

    total_bar_width = .9
    bar_width = total_bar_width / len(dists)

    fig, ax = plt.subplots()
    xs = np.arange(num_states)

    for i,dist in enumerate(dists):
        dist_vals = [p for p,state in dist]
        bar_offset = (i + .5) * bar_width - total_bar_width / 2
        try:
            ax.bar(xs + bar_offset, dist_vals, bar_width, label=labels[i])
        except:
            ax.bar(xs + bar_offset, dist_vals, bar_width)

    ax.set_ylabel("Probability")
    ax.set_xticks(xs)
    ax.set_xticklabels(states)

    if labels:
        ax.legend()
    if title:
        ax.set_title(title)

def ensure_consistent_dists(dists):
    """Given a list of distributions, returns a list of distributions
    that have the same states in the same orders. This function adds "missing" states.
    For example inputting `[[(1,"a")], [(1,"b")]]` results in
    `[[(1,"a"), (0,"b")],[(0,"a"), (1,"b")]]`"""
    all_states = set()
    for dist in dists:
        all_states.update(s for _,s in dist)

    ret = []
    for dist in dists:
        new_dist = []
        dist_hash = {s:v for v,s in dist}
        for state in sorted(all_states):
            if state in dist_hash:
                new_dist.append((dist_hash[state], state))
            else:
                new_dist.append((0, state))
        ret.append(new_dist)
    return ret
```

In [3]:
```
def pick_random(l):
    # Return the state that has the matching probability of randomly generated num that is in [0,
    # For instance, distribution = [(1/4, "a"), (3/4, "b")], then if num is in [0, 1/4], return "
    # if num is in [1/4, 1], return "b".
    num = np.random.rand()
    state_prob_interval = 0
    for i in range(len(l)):
        state_prob, state = l[i]
        state_prob_interval += state_prob # accumluated the state_prob
        if num <= state_prob_interval:
            return state

distribution = [(1/4, "a"), (3/4, "b")]
picks = []
for _ in range(10):
    picks.append(pick_random(distribution))

print("Picking from the distribution", distribution, "10 times gives", picks)
```

```
Picking from the distribution [(0.25, 'a'), (0.75, 'b')] 10 times gives ['a', 'b', 'b', 'b',
'a', 'a', 'b', 'b', 'b', 'b']
```

In [4]:
```
N=100

def pick_random_n(dist, n=100):
    return [pick_random(dist) for i in range(n)] # append list with n sample points from dist

distribution = [(1/4, "a"), (3/4, "b")]
counts = Counter(pick_random_n(distribution, N))

states = [s for (_,s) in distribution]
empirical_dist = [(counts[s]/N, s) for s in states]

plot_distributions(ensure_consistent_dists([distribution, empirical_dist]),
                   labels=["Theoretical", "Empirical"],
                   title="Theoretical and Empirical Distributions")
```



Some Markov Chains!

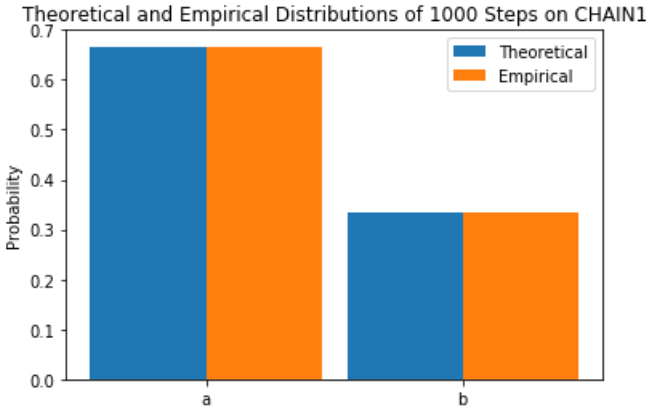
In [5]:
```
#
# These define Markov chains that you will use later
#
CHAIN1 = {
          "a": [(1/2, "a"), (1/2, "b")],
          "b": [(1, "a")]
         }

CHAIN2 = {
          "a": [(.5, "b"), (.5, "e")],
          "b": [(1, "c")],
          "c": [(1, "d")],
          "d": [(1, "a")],
          "e": [(1, "f")],
          "f": [(1, "a")]
         }

CHAIN3 = {
          "a": [(.5, "b"), (.5, "e")],
          "b": [(1, "c")],
          "c": [(1, "d")],
          "d": [(1, "a")],
          "e": [(1, "a")]
         }
```

In [6]:
```
def step(start_state, chain):
    # first we need to get the distribution listfrom start_state
    dist = chain[start_state]
    # we want to return the next state randomly, so let's use pick_random
    return pick_random(dist)

def n_orbit(start_state, chain, n=5):
    # we need to repeat n steps in the markhov chain
    curr_state = start_state
    result = [curr_state] # have n+1 states at the end
    for i in range(n):
        curr_state = step(curr_state, chain)
        result.append(curr_state)
    return result

print("Starting at \"a\" in CHAIN1 and stepping once we end up at", step("a", CHAIN1), ".",
      "If we step 10 times, a realization is", n_orbit("a", CHAIN1, 10))
```

```
Starting at "a" in CHAIN1 and stepping once we end up at b . If we step 10 times, a realization
is ['a', 'a', 'a', 'b', 'a', 'a', 'a', 'a', 'b', 'a', 'b']
```

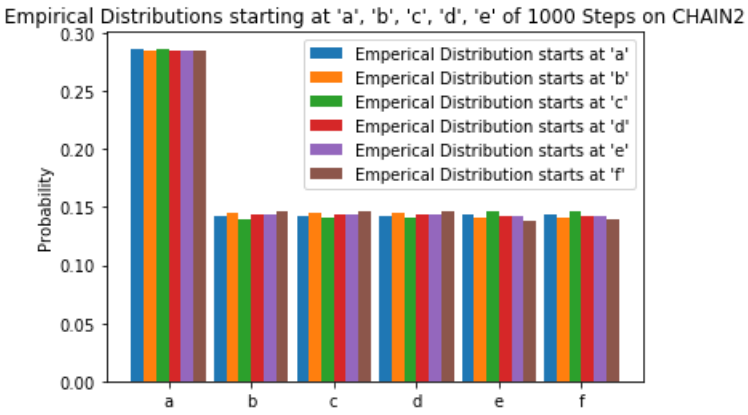
In [7]:
```
def make_dist(realization):
    # Replace this with correc code
    counts_dict = Counter(realization)
    total_steps = sum(counts_dict.values())
    state_list = list(counts_dict.keys())
    state_list.sort() #sort all the states in alphabetical order
    emp_dist = []
    # append relative proportions of the states in the sequence
    for state in state_list:
        emp_dist.append((counts_dict[state]/total_steps, state))

    return emp_dist

empirical_dist = make_dist(n_orbit("a", CHAIN1, 10000))

print("The emperical distribution of the first 1000 steps along CHAIN1 is", empirical_dist)

plot_distributions(ensure_consistent_dists([[(2/3, "a"), (1/3, "b")], empirical_dist]),
                   labels=["Theoretical", "Empirical"],
                   title="Theoretical and Empirical Distributions of 1000 Steps on CHAIN1")
```

```
The emperical distribution of the first 1000 steps along CHAIN1 is [(0.6647335266473353, 'a'),
(0.33526647335266474, 'b')]
```



In [8]:
```
#
# Plot the empirical distributions arising from CHAIN2 starting at "a", "b", ...
#
# Empricial distriutions arising from CHAIN2 starting at "a"
empirical_dist_a = make_dist(n_orbit("a", CHAIN2, 10000))
print("The emperical distribution starting at 'a' of the first 1000 steps along CHAIN2 is", empir

# Empricial distriutions arising from CHAIN2 starting at "b"
empirical_dist_b = make_dist(n_orbit("b", CHAIN2, 10000))
print("The emperical distribution starting at 'b' of the first 1000 steps along CHAIN2 is", empir

# Empricial distriutions arising from CHAIN2 starting at "c"
empirical_dist_c = make_dist(n_orbit("c", CHAIN2, 10000))
print("The emperical distribution starting at 'c' of the first 1000 steps along CHAIN2 is", empir

# Empricial distriutions arising from CHAIN2 starting at "d"
empirical_dist_d = make_dist(n_orbit("d", CHAIN2, 10000))
print("The emperical distribution starting at 'd' of the first 1000 steps along CHAIN2 is", empir

# Empricial distriutions arising from CHAIN2 starting at "e"
empirical_dist_e = make_dist(n_orbit("e", CHAIN2, 10000))
print("The emperical distribution starting at 'e' of the first 1000 steps along CHAIN2 is", empir

# Empricial distriutions arising from CHAIN2 starting at "f"
empirical_dist_f = make_dist(n_orbit("f", CHAIN2, 10000))
print("The emperical distribution starting at 'f' of the first 1000 steps along CHAIN2 is", empir

plot_distributions(ensure_consistent_dists([empirical_dist_a, empirical_dist_b, empirical_dist_c,
                                            empirical_dist_d, empirical_dist_e, empirical_dist_f]
                   labels=["Emperical Distribution starts at 'a'", "Emperical Distribution starts
                           "Emperical Distribution starts at 'c'", "Emperical Distribution starts
                           "Emperical Distribution starts at 'e'", "Emperical Distribution starts
                   title="Empirical Distributions starting at 'a', 'b', 'c', 'd', 'e' of 1000 Ste
```

```
The emperical distribution starting at 'a' of the first 1000 steps along CHAIN2 is [(0.28577142
285771423, 'a'), (0.14278572142785723, 'b'), (0.14278572142785723, 'c'), (0.1426857314268573,
'd'), (0.142985701429857, 'e'), (0.142985701429857, 'f')]
The emperical distribution starting at 'b' of the first 1000 steps along CHAIN2 is [(0.28507149
28507149, 'a'), (0.1446855314468553, 'b'), (0.1446855314468553, 'c'), (0.14458554144585542,
'd'), (0.1404859514048595, 'e'), (0.1404859514048595, 'f')]
The emperical distribution starting at 'c' of the first 1000 steps along CHAIN2 is [(0.28657134
286571345, 'a'), (0.14008599140085992, 'b'), (0.1401859814018598, 'c'), (0.1401859814018598,
'd'), (0.1464853514648535, 'e'), (0.1464853514648535, 'f')]
The emperical distribution starting at 'd' of the first 1000 steps along CHAIN2 is [(0.28547145
285471454, 'a'), (0.1436856314368563, 'b'), (0.1436856314368563, 'c'), (0.14378562143785623,
'd'), (0.1416858314168583, 'e'), (0.1416858314168583, 'f')]
The emperical distribution starting at 'e' of the first 1000 steps along CHAIN2 is [(0.28537146
28537146, 'a'), (0.1436856314368563, 'b'), (0.1436856314368563, 'c'), (0.1436856314368563,
'd'), (0.14178582141785823, 'e'), (0.14178582141785823, 'f')]
The emperical distribution starting at 'f' of the first 1000 steps along CHAIN2 is [(0.28467153
28467153, 'a'), (0.14588541145885411, 'b'), (0.14588541145885411, 'c'), (0.14588541145885411,
'd'), (0.13878612138786123, 'e'), (0.13888611138886112, 'f')]
```



As above numerical data and the graph of comparasion, we can conclude that the starting state does not affect the empirical

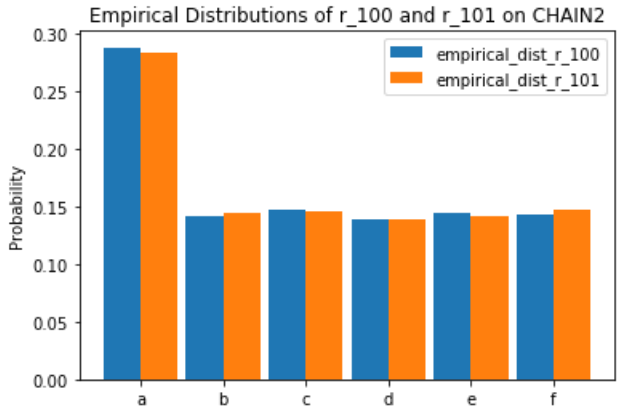
distribution for CHAIN2.

In [9]:
```
#
# Compute the stationary distribution for CHAIN2
#
from numpy import linalg as LA

# first find out the stochastic transition matrix P for CHAIN2
P = np.array([[0, 0, 0, 1, 0, 1], [1/2, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0, 0], [1/2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0]])
# find the eigenvalues 'w' and eigenvectors 'v' for P
w, v = LA.eig(P)
print("The eigenvalues of P are", w)
print("The normalized eigenvectors of P are", v)

# We want to find the normalized eigenvector of P from 'v' that has eigenvalue 1
# so it is the stationary distribution
# we know the column v[:,i] is the eigenvector corresponding to the eigenvalue w[i], so
v_1 = [0.666666667, 0.333333333, 0.333333333, 0.333333333, 0.333333333, 0.333333333]

# The relative porportion does match!
print("The stationary distribution", v_1)
print("agrees with the ratio of the emperical distribution",
      empirical_dist_a)
```

```
The eigenvalues of P are [ 1.        +0.j         -0.17610056+0.86071662j -0.17610056-0.8607166
2j
 -0.64779887+0.j          0.        +0.j          0.        +0.j         ]
The normalized eigenvectors of P are [[ 6.66666667e-01+0.j          5.74411568e-01+0.j
   5.74411568e-01-0.j         -3.44750725e-01+0.j
   3.33066907e-16+0.j          3.33066907e-16+0.j        ]
 [ 3.33333333e-01+0.j         -6.55275775e-02-0.32027538j
  -6.55275775e-02+0.32027538j  2.66093953e-01+0.j
  -5.55111512e-17+0.j         -5.55111512e-17+0.j        ]
 [ 3.33333333e-01+0.j         -3.42202212e-01+0.14614517j
  -3.42202212e-01-0.14614517j -4.10766311e-01+0.j
   5.55111512e-17+0.j          5.55111512e-17+0.j        ]
 [ 3.33333333e-01+0.j          2.41048011e-01+0.34826041j
   2.41048011e-01-0.34826041j  6.34095442e-01+0.j
  -7.07106781e-01+0.j         -7.07106781e-01+0.j        ]
 [ 3.33333333e-01+0.j         -6.55275775e-02-0.32027538j
  -6.55275775e-02+0.32027538j  2.66093953e-01+0.j
  -1.66533454e-16+0.j         -1.66533454e-16+0.j        ]
 [ 3.33333333e-01+0.j         -3.42202212e-01+0.14614517j
  -3.42202212e-01-0.14614517j -4.10766311e-01+0.j
   7.07106781e-01+0.j          7.07106781e-01+0.j        ]]
The stationary distribution [0.666666667, 0.333333333, 0.333333333, 0.333333333, 0.333333333,
0.333333333]
agrees with the ratio of the emperical distribution [(0.28577142285771423, 'a'), (0.14278572142
785723, 'b'), (0.14278572142785723, 'c'), (0.1426857314268573, 'd'), (0.142985701429857, 'e'),
(0.142985701429857, 'f')]
```

In [10]:
```
#
# Plot the distribution of r_100's and r_101's for CHAIN2
#
# some helper functions
def state_at_n(start_state, chain, n=5):
    """This function returns the state after n steps"""
    # create the realization list with n steps, adding the start_state, have n+1 states
    realization = n_orbit(start_state, chain, n)
    # return the state after n steps, i.e. the last state in the realization, and python starts i
    return realization[n]


def generate_r_n(start_state, chain, n, k):
    """This function returns a list of r_n that contains k of them from different realizations"""
    list_of_r_n = []
    for i in range(k):
        list_of_r_n.append(state_at_n(start_state, chain, n))
    return list_of_r_n

# from part a we know that the start state does not affect the realization over time
# so we can use 'a', 'b', or... as the start state
k = 10000 # number of realizations of CHAIN2
list_of_r_100 = generate_r_n('a', CHAIN2, 100, k)
list_of_r_101 = generate_r_n('b', CHAIN2, 101, k)

# let's plot them:
empirical_dist_r_100 = make_dist(list_of_r_100)
print("The emperical distribution of the different r_100's is", empirical_dist_r_100)
empirical_dist_r_101 = make_dist(list_of_r_101)
print("The emperical distribution of the different r_101's is", empirical_dist_r_101)

plot_distributions(ensure_consistent_dists([empirical_dist_r_100, empirical_dist_r_101]),
                   labels=["empirical_dist_r_100", "empirical_dist_r_101"],
                   title="Empirical Distributions of r_100 and r_101 on CHAIN2")
```

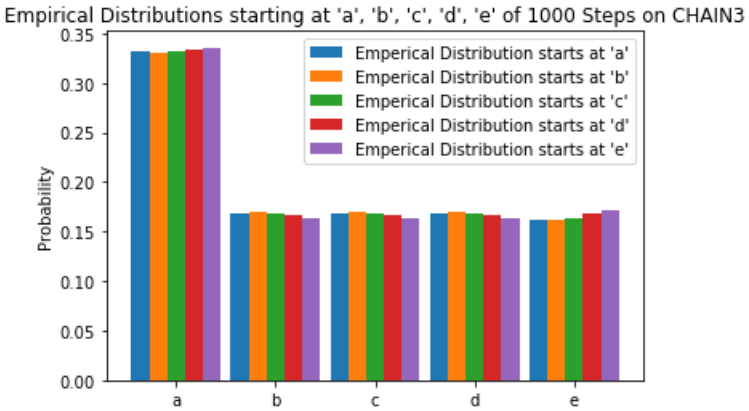
```
The emperical distribution of the different r_100's is [(0.2876, 'a'), (0.1411, 'b'), (0.1465,
'c'), (0.1386, 'd'), (0.1437, 'e'), (0.1425, 'f')]
The emperical distribution of the different r_101's is [(0.2827, 'a'), (0.1447, 'b'), (0.145,
'c'), (0.1392, 'd'), (0.1418, 'e'), (0.1466, 'f')]
```



This is the same from the stationary distribution (within two decimal places) for CHAIN2, as expected, since all empricial distribution should eventually converge to the stationary distribution, if there is a stationary distribution. We know CHAIN2 can be modeled by T and has a stationary distribution.

In [11]:
```
#
# Plot the empirical distributions arising from CHAIN3 starting at "a", "b", ...
#

# Empricial distriutions arising from CHAIN3 starting at "a"
empirical_dist_a3 = make_dist(n_orbit("a", CHAIN3, 10000))
print("The emperical distribution starting at 'a' of the first 1000 steps along CHAIN3 is", empir

# Empricial distriutions arising from CHAIN3 starting at "b"
empirical_dist_b3 = make_dist(n_orbit("b", CHAIN3, 10000))
print("The emperical distribution starting at 'b' of the first 1000 steps along CHAIN3 is", empir

# Empricial distriutions arising from CHAIN3 starting at "c"
empirical_dist_c3 = make_dist(n_orbit("c", CHAIN3, 10000))
print("The emperical distribution starting at 'c' of the first 1000 steps along CHAIN3 is", empir

# Empricial distriutions arising from CHAIN3 starting at "d"
empirical_dist_d3 = make_dist(n_orbit("d", CHAIN3, 10000))
print("The emperical distribution starting at 'd' of the first 1000 steps along CHAIN3 is", empir

# Empricial distriutions arising from CHAIN3 starting at "e"
empirical_dist_e3 = make_dist(n_orbit("e", CHAIN3, 10000))
print("The emperical distribution starting at 'e' of the first 1000 steps along CHAIN3 is", empir


plot_distributions(ensure_consistent_dists([empirical_dist_a3, empirical_dist_b3, empirical_dist_
                                            empirical_dist_d3, empirical_dist_e3]),
                   labels=["Emperical Distribution starts at 'a'", "Emperical Distribution starts
                           "Emperical Distribution starts at 'c'", "Emperical Distribution starts
                           "Emperical Distribution starts at 'e'"],
                   title="Empirical Distributions starting at 'a', 'b', 'c', 'd', 'e' of 1000 Ste
```

```
The emperical distribution starting at 'a' of the first 1000 steps along CHAIN3 is [(0.33126687
331266874, 'a'), (0.16878312168783122, 'b'), (0.16878312168783122, 'c'), (0.16878312168783122,
'd'), (0.1623837616238376, 'e')]
The emperical distribution starting at 'b' of the first 1000 steps along CHAIN3 is [(0.33076692
33076692, 'a'), (0.16918308169183083, 'b'), (0.16918308169183083, 'c'), (0.16918308169183083,
'd'), (0.1616838316168383, 'e')]
The emperical distribution starting at 'c' of the first 1000 steps along CHAIN3 is [(0.33166683
331666835, 'a'), (0.16828317168283172, 'b'), (0.1683831616838316, 'c'), (0.1683831616838316,
'd'), (0.16328367163283672, 'e')]
The emperical distribution starting at 'd' of the first 1000 steps along CHAIN3 is [(0.33376662
33376662, 'a'), (0.16618338166183383, 'b'), (0.16618338166183383, 'c'), (0.16628337166283372,
'd'), (0.1675832416758324, 'e')]
The emperical distribution starting at 'e' of the first 1000 steps along CHAIN3 is [(0.33576642
33576642, 'a'), (0.16418358164183583, 'b'), (0.16418358164183583, 'c'), (0.16418358164183583,
'd'), (0.17168283171682833, 'e')]
```



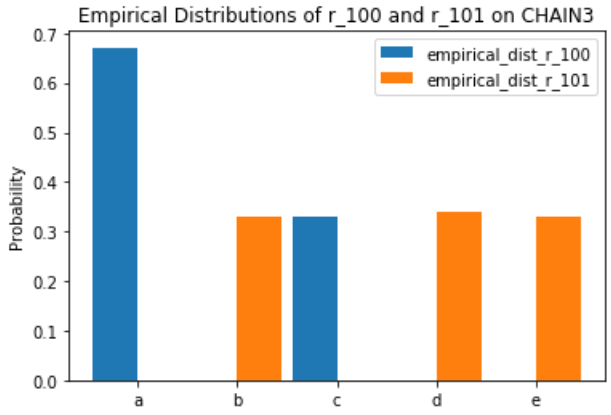
As above numerical data and the graph of comparasion, we can conclude that the starting state does not affect the empirical distribution for CHAIN3.

In [12]:
```
#
# Compute the stationary distribution for CHAIN3
#

# first find out the stochastic transition matrix P2 for CHAIN3
P2 = np.array([[0, 0, 0, 1, 1], [1/2, 0, 0, 0, 0], [0, 1, 0, 0, 0],
               [0, 0, 1, 0, 0], [1/2, 0, 0, 0, 0]])
# find the eigenvalues 'w' and eigenvectors 'v' for P
w2, v2 = LA.eig(P2)
print("The eigenvalues of P2 are", w2)
print("The normalized eigenvectors of P2 are", v2)

# We want to find the normalized eigenvector of P2 from 'v2' that has eigenvalue 1
# so it is the stationary distribution
# we know the column v2[:,i] is the eigenvector corresponding to the eigenvalue w2[i], so
v2_1 = [-0.707106781, -0.353553391, -0.353553391, -0.353553391, -0.353553391]
# unfortuntaetly this is negative, but we know any non-zero scalar mutiple of an eigenvector is a
# simply mutiple v2_1 by -1, we get the stationary distribution of CHAIN3, as needed
v2_1_dist = [abs(entry) for entry in v2_1]

# The relative porportion does match!
print("The stationary distribution", v2_1_dist)
print("agrees with the ratio of the emperical distribution",
      empirical_dist_a3)
```

```
The eigenvalues of P2 are [-1.00000000e+00+0.j          1.00000000e+00+0.j
 -5.20417043e-17+0.70710678j -5.20417043e-17-0.70710678j
  0.00000000e+00+0.j        ]
The normalized eigenvectors of P2 are [[ 7.07106781e-01+0.00000000e+00j -7.07106781e-01+0.00000
000e+00j
   1.34205714e-16-4.47213595e-01j  1.34205714e-16+4.47213595e-01j
  -5.03570533e-18+0.00000000e+00j]
 [-3.53553391e-01+0.00000000e+00j -3.53553391e-01+0.00000000e+00j
  -3.16227766e-01+5.28813051e-19j -3.16227766e-01-5.28813051e-19j
  -5.55111512e-17+0.00000000e+00j]
 [ 3.53553391e-01+0.00000000e+00j -3.53553391e-01+0.00000000e+00j
  -1.63175243e-16+4.47213595e-01j -1.63175243e-16-4.47213595e-01j
  -1.87446841e-17+0.00000000e+00j]
 [-3.53553391e-01+0.00000000e+00j -3.53553391e-01+0.00000000e+00j
   6.32455532e-01+0.00000000e+00j  6.32455532e-01-0.00000000e+00j
  -7.07106781e-01+0.00000000e+00j]
 [-3.53553391e-01+0.00000000e+00j -3.53553391e-01+0.00000000e+00j
  -3.16227766e-01-2.20281895e-17j -3.16227766e-01+2.20281895e-17j
   7.07106781e-01+0.00000000e+00j]]
The stationary distribution [0.707106781, 0.353553391, 0.353553391, 0.353553391, 0.353553391]
agrees with the ratio of the emperical distribution [(0.33126687331266874, 'a'), (0.16878312168
783122, 'b'), (0.16878312168783122, 'c'), (0.16878312168783122, 'd'), (0.1623837616238376,
'e')]
```

In [15]:
```
#
# Plot the distribution of r_100's and r_101's for CHAIN3
#
# from part a we know that the start state does not affect the realization over time
# so we can use 'a', 'b', or... as the start state
k = 10000 # number of realizations of CHAIN3
list_of_r_100_chain3 = generate_r_n('a', CHAIN3, 100, k)
list_of_r_101_chain3 = generate_r_n('a', CHAIN3, 101, k)

# let's plot them:
empirical_dist_r_100_chain3 = make_dist(list_of_r_100_chain3)
print("The emperical distribution of the different r_100's is", empirical_dist_r_100_chain3)
empirical_dist_r_101_chain3 = make_dist(list_of_r_101_chain3)
print("The emperical distribution of the different r_101's is", empirical_dist_r_101_chain3)

plot_distributions(ensure_consistent_dists([empirical_dist_r_100_chain3, empirical_dist_r_101_cha
                   labels=["empirical_dist_r_100", "empirical_dist_r_101"],
                   title="Empirical Distributions of r_100 and r_101 on CHAIN3")
```

```
The emperical distribution of the different r_100's is [(0.6711, 'a'), (0.3289, 'c')]
The emperical distribution of the different r_101's is [(0.3299, 'b'), (0.341, 'd'), (0.3291,
'e')]
```



The result is different than expected as even and odd steps r's matter. But if you take a look as a whole, they do fall into the empricial distribution.

Simulate Speech

In [16]:
```
#
# Some useful functions for text analysis
#
def get_words(raw_text):
    """Given a text string, remove all punctuation and capitalization
    and return a list of words."""
    import re
    return re.sub(r"\s+", " ", re.sub(r"[\W\d]", " ", raw_text)).lower().strip().split(" ")

#
# Download the complete works of Shakespeare and the Ontario Criminal Code
#
import requests
response = requests.get("https://github.com/siefkenj/2020-MAT-335-webpage/raw/master/homework/sha
raw_text = response.text
SHAKE = get_words(raw_text)

response = requests.get("https://github.com/siefkenj/2020-MAT-335-webpage/raw/master/homework/ont
raw_text = response.text
CRIMINAL = get_words(raw_text)
```

In [17]:
```
def make_chain(words):
    # create a dict that stores all the words and how many times it gets to the next word
    counts = {}
    for i in range(len(words)-1):
        if words[i] in counts:
            if words[i+1] not in counts[words[i]]: # first apperance of this transition state, cr
                counts[words[i]][words[i+1]] = 1
            else: # appeared before, add one more count
                counts[words[i]][words[i+1]] += 1
        else: # never step to this state, create new state
            counts[words[i]] = {}
            counts[words[i]][words[i+1]] = 1

    # create the new dict that change counts to have the transition probability
    # by dividing each counts of the sum of all the counts,'count_sum' under one key of <dict> co
    result = {}
    for word in counts:
        result[word] = [] # create the new list of transition probability to result

        count_sum = 0
        for state in counts[word]: # first sum all the counts under one <key> word
            count_sum += counts[word][state]

        for state in counts[word]: # then update that word's count to transition probability
            new_count = counts[word][state]/count_sum
            result[word].append((new_count, state))

    return result

print("Simulating CHAIN1 for 1000 steps and then using the simulation to recover",
      "a Markov chain gives:",
      make_chain(n_orbit("a", CHAIN1, 1000)))
```

```
Simulating CHAIN1 for 1000 steps and then using the simulation to recover a Markov chain gives:
{'a': [(0.4727540500736377, 'b'), (0.5272459499263623, 'a')], 'b': [(1.0, 'a')]}
```

In [18]:
```
SHAKE_CHAIN = make_chain(SHAKE)
CRIMINAL_CHAIN = make_chain(CRIMINAL)
```

In [19]:
```
#
# A realization of Shakespeare starting words “the”
#
real_1 = " ".join(n_orbit("the", SHAKE_CHAIN, 30))
print("A realization of Shakespeare starting words “the” is:", real_1, "\n")

# A realization of Shakespeare starting words “sunset”
real_2 = " ".join(n_orbit("sunset", SHAKE_CHAIN, 30))
print("A realization of Shakespeare starting words “sunset” is:", real_2, "\n")

# A realization of Shakespeare starting words “satisfied"
real_3 = " ".join(n_orbit("satisfied", SHAKE_CHAIN, 30))
print("A realization of Shakespeare starting words “satisfied” is:", real_3, "\n")
```

```
A realization of Shakespeare starting words “the” is: the gloss on my lord enter flavius you ar
e great a further trial day and her two o ersways the word o to trouble thee in the time fearef
ull consumers you

A realization of Shakespeare starting words “sunset” is: sunset fadeth in scorn of fate hies an
d slaughter d let me i speak of cuckolds first watch d his wife and precious friends cloten tho
u like a punk is my

A realization of Shakespeare starting words “satisfied” is: satisfied i am not he ne er stained
with my meaning is slippery if thou not stand on and though it rosalind that the child lies i s
py advantages enter bassanio
```

Make the realization of Shakespeare sounds like a sentence spoken by a human:

```
"The other times worcester and trumpet send me untir d, our well aveng d, as if he loved
 your majesty. And reason hated so far, i never have them make way."

"Sunset set of mutton or i ken the one appearing to wish. He left solely. A king lies thy
plainness. Do me othello she heir? Am possess you, madam I cleave."

"Satisfied exeunt musicians attending star. The top curling their cheeks, gloucester edga
r armed commons within this noble. Sufferance comes blubber d exit. Hortensio trow to sta
y d stool and aim."
```

In [20]:
```
#
# A realization of the Ontario Criminal Code starting words “the”
#
real_4 = " ".join(n_orbit("the", CRIMINAL_CHAIN, 30))
print("A realization of the Ontario Criminal Code starting words “the” is:", real_4, "\n")

# A realization of the Ontario Criminal Code starting words “sunset”
real_5 = " ".join(n_orbit("sunset", CRIMINAL_CHAIN, 30))
print("A realization of the Ontario Criminal Code starting words “sunset” is:", real_5, "\n")

# A realization of the Ontario Criminal Code starting words “satisfied"
real_6 = " ".join(n_orbit("satisfied", CRIMINAL_CHAIN, 30))
print("A realization of the Ontario Criminal Code starting words “satisfied” is:", real_6, "\n")
```

```
A realization of the Ontario Criminal Code starting words “the” is: the revised statutes of tri
al process against the prosecutor and more than years or in section marginal note attorney gene
ral or against this act alleged to be impracticable for unlawful act

A realization of the Ontario Criminal Code starting words “sunset” is: sunset provision of the
other than a conveyance a person in relation to any stage of the following things connected wit
h a summons in addition the public good behaviour in section

A realization of the Ontario Criminal Code starting words “satisfied” is: satisfied by way if a
legal assistance in canada ii the court has been issued and appears to the indictment nunavut o
rders otherwise be paid in any other punishment of foreign
```

Make the realizationo of the Ontario Criminal Code sounds like a sentence spoken by a human:

```
"The appellant shall give reasons why an offence is satisfied that, by a period of every
 one or ii the preliminary inquiry, c s c any act or receives royal Canadian."

"Sunset provision without excuse procures or c s c st supp s previous version marginal. N
ote: Time in or concealing person as authorized by the birth of a judge means the."

"Satisfied that there was laid if previously reviewed under paragraph, a bodily substance
s, etc. of the applicant has been taken under section or conveyed whether with a person n
ot a ii."
```

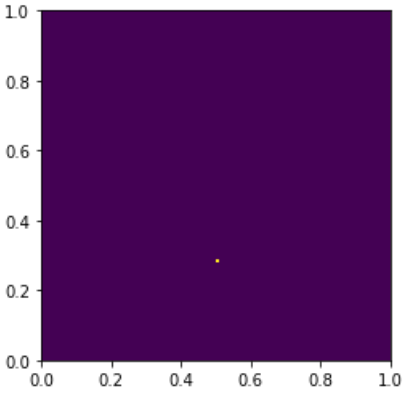
Iterated Functions

In [21]:
```
#
# Some helpful functions
#

def apply(point, funcs):
    """Apply a sequence of functions to a point."""
    for f in funcs:
        point = f(point)
    return point

def render_points_to_array(points, array, extent=[0, 1, 0, 1], additive=False):
    """Given a list of points `points` and a 2d numpy array `array`,
    "draw" the points to the array. The resulting array is suitable for displaying
    with `imshow`. """

    array = array.copy()
    h, w = array.shape

    for p in points:
        # conver the xy-coordinates to array indices
        x = np.clip(int((p[0] - extent[0]) / (extent[1] - extent[0]) * w), 0, w - 1)
        y = np.clip(int((p[1] - extent[2]) / (extent[3] - extent[2]) * h), 0, h - 1)
        if additive:
            array[y][x] += 1
        else:
            array[y][x] = 1
    return array
```

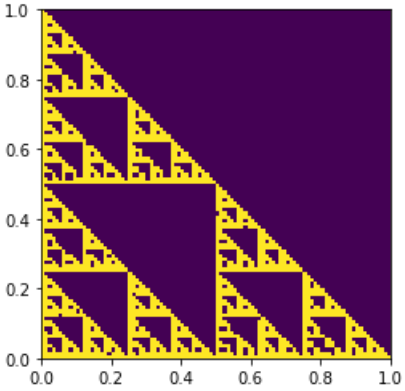
In [22]:
```
#
# Functions for our first iterated function system!
#
def f1(p):
    return (p/2)
def f2(p):
    return (p/2 + np.array([1/2,0]))
def f3(p):
    return p/2 + np.array([0,1/2])

F_CHAIN = {
    "start": [(1/3, f1), (1/3, f2), (1/3, f3)],
    f1: [(1/3, f1), (1/3, f2), (1/3, f3)],
    f2: [(1/3, f1), (1/3, f2), (1/3, f3)],
    f3: [(1/3, f1), (1/3, f2), (1/3, f3)],
    }
```

In [23]:
```
N = 100
zs = np.zeros((N,N))

funcs = n_orbit("start", F_CHAIN, 10)
# The first state in this realization is "start", not actually a function,
# so we will remove that with an *array slice*
funcs = funcs[1:]

# Apply the sequence of functions to the starting point (1,1)
sample_point = apply(np.array([1,1]), funcs)

# .__name__ is Python magic to get the name of a function as a string
print("Applying the sequence of functions", [f.__name__ for f in funcs], "to (1,1) gives", sample

fig, ax = plt.subplots()
extent = [0, 1, 0, 1]
# Plot our lonely sample point!
ax.imshow(render_points_to_array([sample_point], zs, extent), cmap="viridis", extent=extent, orig

plt.show()
```

```
Applying the sequence of functions ['f3', 'f3', 'f3', 'f1', 'f1', 'f3', 'f1', 'f1', 'f3', 'f2']
to (1,1) gives [0.50097656 0.2890625 ]
```

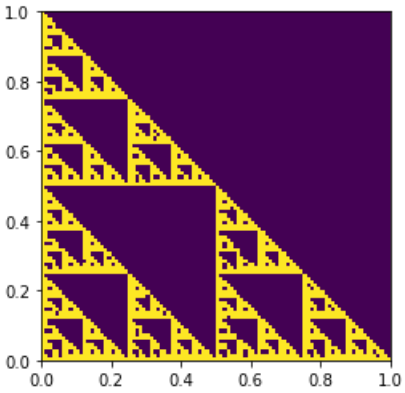


In [24]:
```
#
# Plot the distribution produced by F_CHAIN
#

N = 100
zs = np.zeros((N,N))


# Apply the sequence of functions generated by F_CHAIN to random points, create a list of sample
def sample_points(chain, state_state, N):
    result = []
    for i in range(N):
        funcs = n_orbit(state_state, chain, 100)
        point = apply(np.array([np.random.rand(), np.random.rand()]), funcs[1:])
        result.append(point)
    return result

fig, ax = plt.subplots()
extent = [0, 1, 0, 1]

ax.imshow(render_points_to_array(sample_points(F_CHAIN, "start", 10000), zs, extent), cmap="virid

plt.show()
```

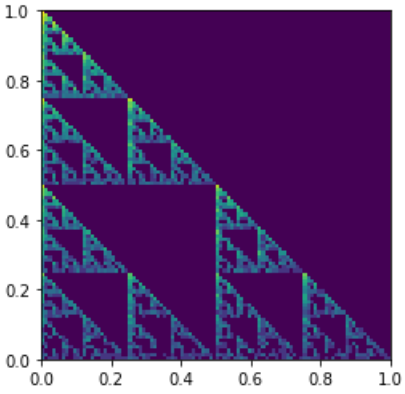


This looks very familar... It is the slanted sierpinski trangle, which we have seen in class!

In [25]:
```
#
# Create F_CHAIN2 and plot the densities of both F_CHAIN and F_CHAIN2
#

F_CHAIN2 = {
    "start": [(1/4, f1), (1/4, f2), (1/2, f3)],
    f1: [(1/4, f1), (1/4, f2), (1/2, f3)],
    f2: [(1/4, f1), (1/4, f2), (1/2, f3)],
    f3: [(1/4, f1), (1/4, f2), (1/2, f3)],
    }

# Plot F_CHAIN densitity
N = 100
zs = np.zeros((N,N))


fig, ax1 = plt.subplots()
extent = [0, 1, 0, 1]

ax1.imshow(np.log(render_points_to_array(sample_points(F_CHAIN, "start", 10000), zs, extent)+1),
          cmap="viridis", extent=extent, origin="lower")

plt.show()
```

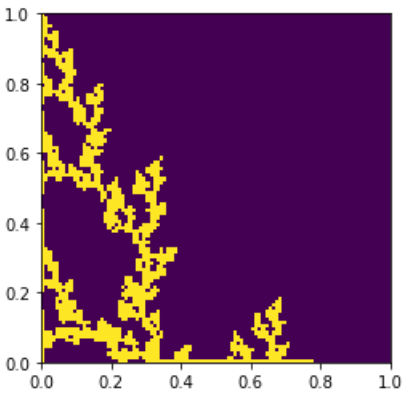


In [26]:
```
# Plot F_CHAIN2 densitity
fig, ax2 = plt.subplots()
ax2.imshow(np.log(render_points_to_array(sample_points(F_CHAIN2, "start", 10000), zs, extent, add
          cmap="viridis", extent=extent, origin="lower")

plt.show()
```



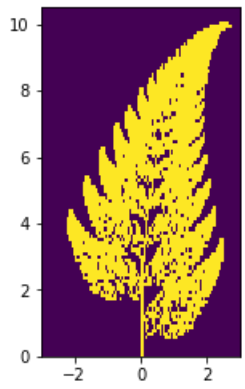
The results are not what I expected, as I thought different transition probabilities of F_CHAIN and F_CHAIN2 would make a difference. But, it is true that overtime it does not matter.

In [27]:
```
#
# Create ROT_CHAIN and plot the resulting distribution
#
def f1(p): # redefine so it rotates CCW 30 degree for the resulting point
    x, y = p/2
    return np.array([3**(1/2)/2 * x - 1/2 * y, 1/2*x + 3**(1/2)/2*y])
def f2(p): # redefine so it rotates CW 30 degree for the resulting point
    x, y = p/2 + np.array([1/2,0])
    return np.array([3**(1/2)/2*x + 1/2*y, -1/2*x + 3**(1/2)/2*y])
def f3(p):
    return p/2 + np.array([0,1/2])

ROT_CHAIN = {
    "start": [(1/3, f1), (1/3, f2), (1/3, f3)],
    f1: [(1/3, f1), (1/3, f2), (1/3, f3)],
    f2: [(1/3, f1), (1/3, f2), (1/3, f3)],
    f3: [(1/3, f1), (1/3, f2), (1/3, f3)],
    }

# Plot ROT_CHAIN densitity
N = 100
zs = np.zeros((N,N))

fig, ax = plt.subplots()
extent = [0, 1, 0, 1]

ax.imshow(render_points_to_array(sample_points(ROT_CHAIN, "start", 10000), zs, extent),
          cmap="viridis", extent=extent, origin="lower")

plt.show()
```



This is an interesting flower looking picture.

In [28]:
```
#
# Create BARNSLEY_CHAIN here
#
def f1(p):
    x, y = p
    return np.array([0, 0.16*y])

def f2(p):
    x, y = p
    return np.array([0.85*x + 0.04*y, -0.04*x + 0.85*y + 1.6])

def f3(p):
    x, y = p
    return np.array([0.2*x - 0.26*y, 0.23*x + 0.22*y + 1.6])

def f4(p):
    x, y = p
    return np.array([-0.15*x + 0.28*y, 0.26*x + 0.24*y + 0.44])

BARNSLEY_CHAIN = {
    "start": [(0.01, f1), (0.85, f2), (0.07, f3), (0.07, f4)],
    f1: [(0.01, f1), (0.85, f2), (0.07, f3), (0.07, f4)],
    f2: [(0.01, f1), (0.85, f2), (0.07, f3), (0.07, f4)],
    f3: [(0.01, f1), (0.85, f2), (0.07, f3), (0.07, f4)],
    f4: [(0.01, f1), (0.85, f2), (0.07, f3), (0.07, f4)],
    }

# Plot BARNSLEY_CHAIN densitity
N = 100
zs = np.zeros((N,N))

fig, ax = plt.subplots()
extent = [-3, 3, 0, 10.5]

ax.imshow(render_points_to_array(sample_points(BARNSLEY_CHAIN, "start", 20000), zs, extent),
          cmap="viridis", extent=extent, origin="lower")

plt.show()
```



In [ ]:
```
```