DYS HW4-1

4.1 Introduction to the Lorenz model

Deadline: 13 Dec 23:59

(2 points)

In[68]:=

The three-dimensional Lorenz flow is given by

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= rx - y - xz \; . \\ \dot{z} &= xy - bz \end{aligned} \qquad (1)$$

The Lorenz system is named after the meteorologist Edward Norton Lorenz who studied it extensively. He found that the system (1) exhibits a fractal attractor for the parameter values $\sigma = 10$, $b = 8/3$ and $r = 28$. This attractor is nowadays called Lorenz attractor.

Out[68]=

4.1 Introduction to the Lorenz model

Deadline: 13 Dec 23:59

(2 points)

The three-dimensional Lorenz flow is given by

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= rx - y - xz \; . \\ \dot{z} &= xy - bz \end{aligned} \qquad (1)$$

The Lorenz system is named after the meteorologist Edward Norton Lorenz who studied it extensively. He found that the system (1) exhibits a fractal attractor for the parameter values $\sigma = 10$, $b = 8/3$ and $r = 28$. This attractor is nowadays called Lorenz attractor.

a.) Fixed points of Lorentz System

(a) How many fixed points does the Lorenz system have, and how many of them are stable for the parameter values given above? Give your answer as the vector [number of fixed points,number of stable fixed points].

In[69]:=
```
ClearAll["Global`*"]

xPrime = s * y - s * x;
yPrime = r * x - y - x * z;
zPrime = x * y - b * z;

s = 10;
b = 8/3;
r = 28;

FP1 = {0,0,0};
FP2 = {Sqrt[72],Sqrt[72],27};
FP3 = {-Sqrt[72],-Sqrt[72],27};


Jacobi = D[{xPrime,yPrime,zPrime},{{x,y,z}}]
JacobMatrix = Jacobi // MatrixForm
Eig = N[Eigenvalues[Jacobi]]

(*Parameters*)
s = 10;
b = 8/3;
r = 28;
x = 0;
y = 0;
z = 0;
Eig = N[Eigenvalues[Jacobi]]

x = Sqrt[72];
y = Sqrt[72];
z = 27;
Eig = N[Eigenvalues[Jacobi]]

x = -Sqrt[72];
y = -Sqrt[72];
z = 27;
Eig = N[Eigenvalues[Jacobi]]
```

Out[79]= $\left\{\{-10, 10, 0\}, \{28 - z, -1, -x\}, \left\{y, x, -\frac{8}{3}\right\}\right\}$

Out[80]//MatrixForm=

$$\begin{pmatrix} -10 & 10 & 0 \\ 28-z & -1 & -x \\ y & x & -\frac{8}{3} \end{pmatrix}$$

Out[81]=
```
{0.333333
  Root[-19440 + 270 x^2 + 270 x y + 720 z + (-2166 + 9 x^2 + 90 z) #1 + 41 #1^2 + #1^3 &, 1],
 0.333333 Root[-19440 + 270 x^2 + 270 x y + 720 z + (-2166 + 9 x^2 + 90 z) #1 + 41 #1^2 + #1^3 &,
   2], 0.333333
  Root[-19440 + 270 x^2 + 270 x y + 720 z + (-2166 + 9 x^2 + 90 z) #1 + 41 #1^2 + #1^3 &, 3]}
```

Out[88]= {-22.8277, 11.8277, -2.66667}

Out[92]= {-13.8546, 0.0939556 + 10.1945 i, 0.0939556 - 10.1945 i}

Out[96]= {-13.8546, 0.0939556 + 10.1945 i, 0.0939556 - 10.1945 i}

In[97]:=

In[98]:=

for FP1 = {-22.8277,11.8277,-2.66667}, for FP2 = {-13.8546,0.0939556+10.1945 *i*,0.0939556-10.1945 *i*}, for FP3 = {-13.8546,0.0939556+10.1945 *i*,0.0939556-10.1945 *i*}

---

b.) Solve system numerically

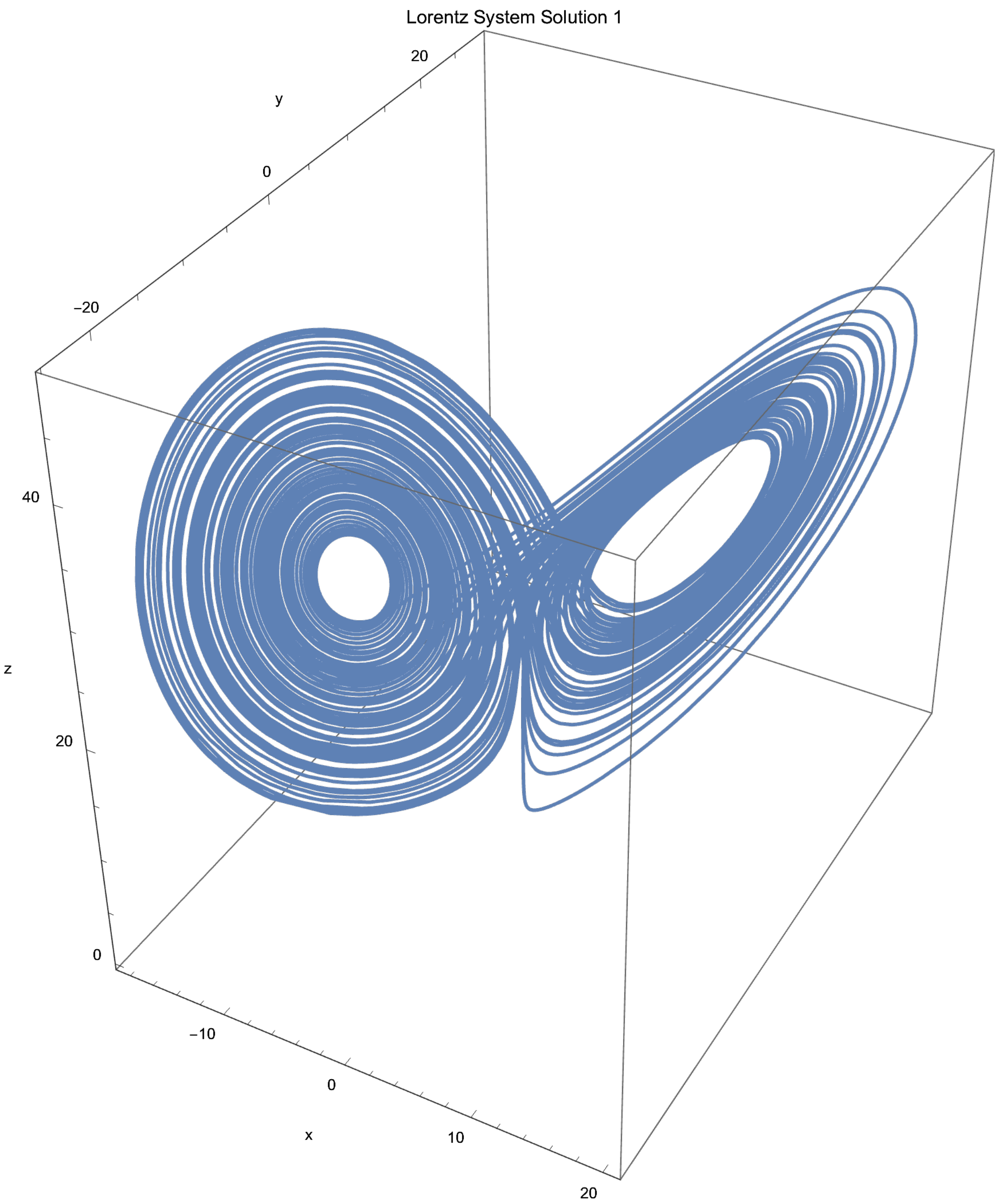
(b) Solve the equations (1) numerically using the parameters stated above for some initial condition close to the origin. Plot an approximation of Lorenz attractor obtained by discarding the initial part of the solution.
Upload your figure as .pdf or .png.

In[99]:=
```
ClearAll["Global`*"]

(*Parameters of Lorentz*)
s = 10;
b = 8/3;
r = 28;

xPrime = s * y - s * x;
yPrime = r * x - y - x * z;
zPrime = x * y - b * z;

system1 = {
    x'[t] == s * y[t] - s * x[t],
    y'[t] == r * x[t] - y[t] - x[t] * z[t],
    z'[t] == x[t] * y[t] - b * z[t]
};

FP1 = {0,0,0};
delta = 0.1;
startingPoint1 = FP1 + delta;

initialConditions1 = {x[0] == startingPoint1[[1]], y[0] == startingPoint1[[2]], z[0] == star
```

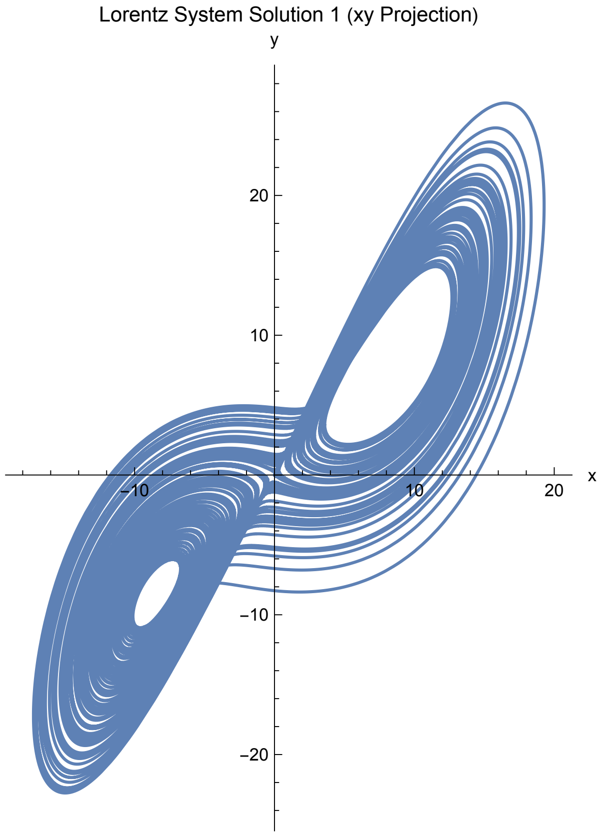
```
t0 = 0;
tMax = 100;
tIntersting = 10;

solution1 = NDSolve[{system1, initialConditions1}, {x,y,z}, {t, t0, tMax}]

(* Define the solution function *)
sol1[t_] = {x[t], y[t], z[t]} /. solution1[[1]];

(* Plot the solution using ParametricPlot3D *)
ParametricPlot3D[sol1[t], {t, tIntersting, tMax}, PlotRange → All, AxesLabel → {"x", "
  PlotLabel → "Lorentz System Solution 1", ImageSize → Large, MaxRecursion → 15]

(* Define the solution functions for each variable *)
xSol[t_] = x[t] /. solution1[[1]];
ySol[t_] = y[t] /. solution1[[1]];
zSol[t_] = z[t] /. solution1[[1]];

(* Plot xy, xz, and yz projections *)
xyPlot = ParametricPlot[{xSol[t], ySol[t]}, {t, tIntersting, tMax},
  PlotRange → All, AxesLabel → {"x", "y"}, MaxRecursion → 15,
  PlotLabel → "Lorentz System Solution 1 (xy Projection)", ImageSize → Medium];

xzPlot = ParametricPlot[{xSol[t], zSol[t]}, {t, tIntersting, tMax},
  PlotRange → All, AxesLabel → {"x", "z"},MaxRecursion → 15,
  PlotLabel → "Lorentz System Solution 1 (xz Projection)", ImageSize → Medium];

yzPlot = ParametricPlot[{ySol[t], zSol[t]}, {t, tIntersting, tMax},
  PlotRange → All, AxesLabel → {"y", "z"}, MaxRecursion → 15,
  PlotLabel → "Lorentz System Solution 1 (yz Projection)", ImageSize → Medium];

(* Show all plots *)
Show[xyPlot]
Show[xzPlot]
Show[yzPlot]
(*
```

```
Out[114]= {{x → InterpolatingFunction[ Domain: {{0., 100. }} Output: scalar ],
  y → InterpolatingFunction[ Domain: {{0., 100. }} Output: scalar ],
  z → InterpolatingFunction[ Domain: {{0., 100. }} Output: scalar ]}}
```

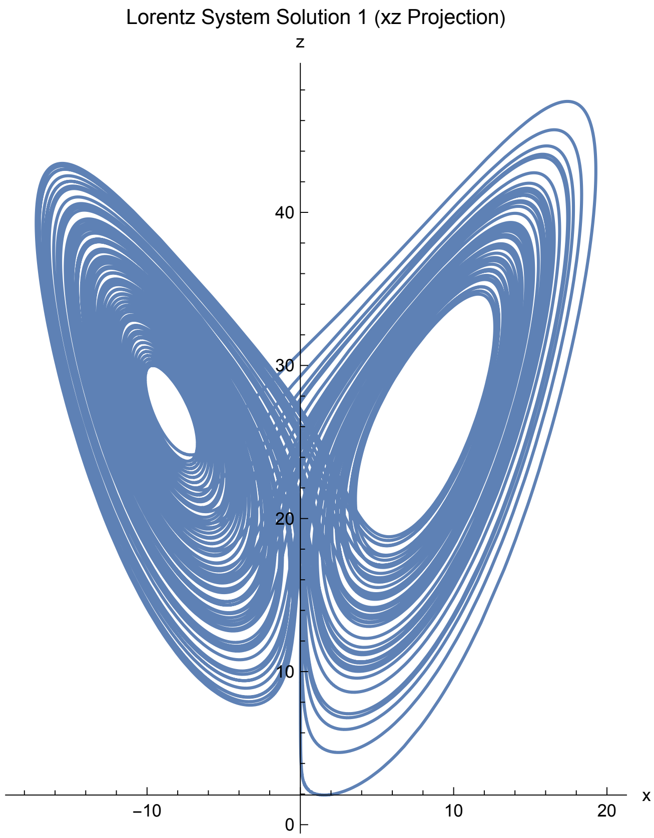
Out[116]=



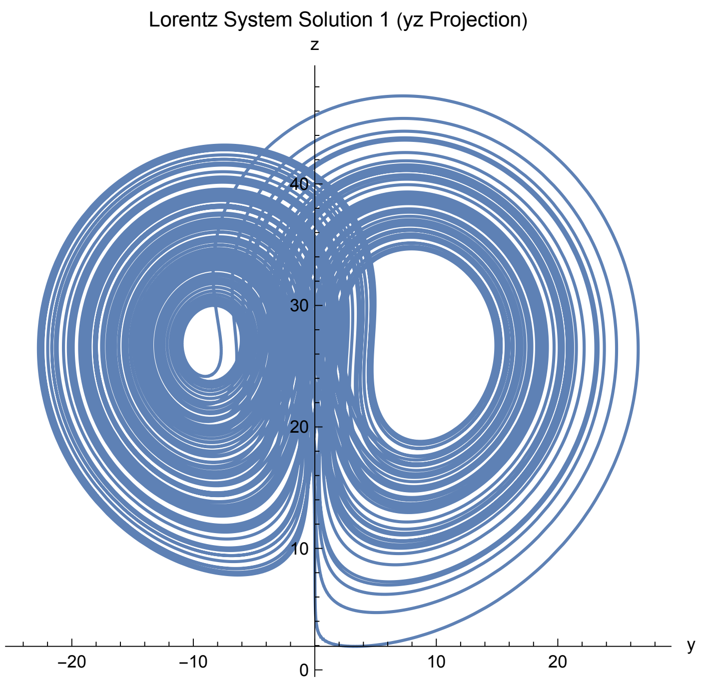
Out[123]=



Out[124]=



Out[125]=



In[126]:=

c.) Compute the stability matrix J

In[127]:=

(c) Compute the stability matrix $\mathbb{J}_{ij} = \partial F_i / \partial x_j$ of the flow (1). Give your result as the matrix $[[J_{11}, J_{12}, J_{13}], [J_{21}, J_{22}, J_{23}], [J_{31}, J_{32}, J_{33}]]$.

(in terms of sigma,r,z,x,y,b)

A A A

Out[127]=

(c) Compute the stability matrix $\mathbb{J}_{ij} = \partial F_i / \partial x_j$ of the flow (1). Give your result as the matrix $[[J_{11}, J_{12}, J_{13}], [J_{21}, J_{22}, J_{23}], [J_{31}, J_{32}, J_{33}]]$.

(in terms of sigma,r,z,x,y,b)

A A A

In[128]:=

```
(* Analog a.) *)
```

d.) Trace of the stability matrix and sum of the Lyapunov Exponents

In[129]:=

```
ClearAll["Global`*"]

xPrime = s * y - s * x;
yPrime = r * x - y - x * z;
zPrime = x * y - b * z;

(*s = 10;
b = 8/3;
r = 28;*)

FP1 = {0,0,0};
FP2 = {Sqrt[72],Sqrt[72],27};
FP3 = {-Sqrt[72],-Sqrt[72],27};

Jacobi = D[{xPrime,yPrime,zPrime},{{x,y,z}}]

traceJ = Tr[Jacobi]
eigVl = Eigenvalues[Jacobi];
sumOfEigVl = Total[eigVl] // Simplify
```

Out[136]= {{-s, s, 0}, {r - z, -1, -x}, {y, x, -b}}

Out[137]= -1 - b - s

Out[139]= -1 - b - s