

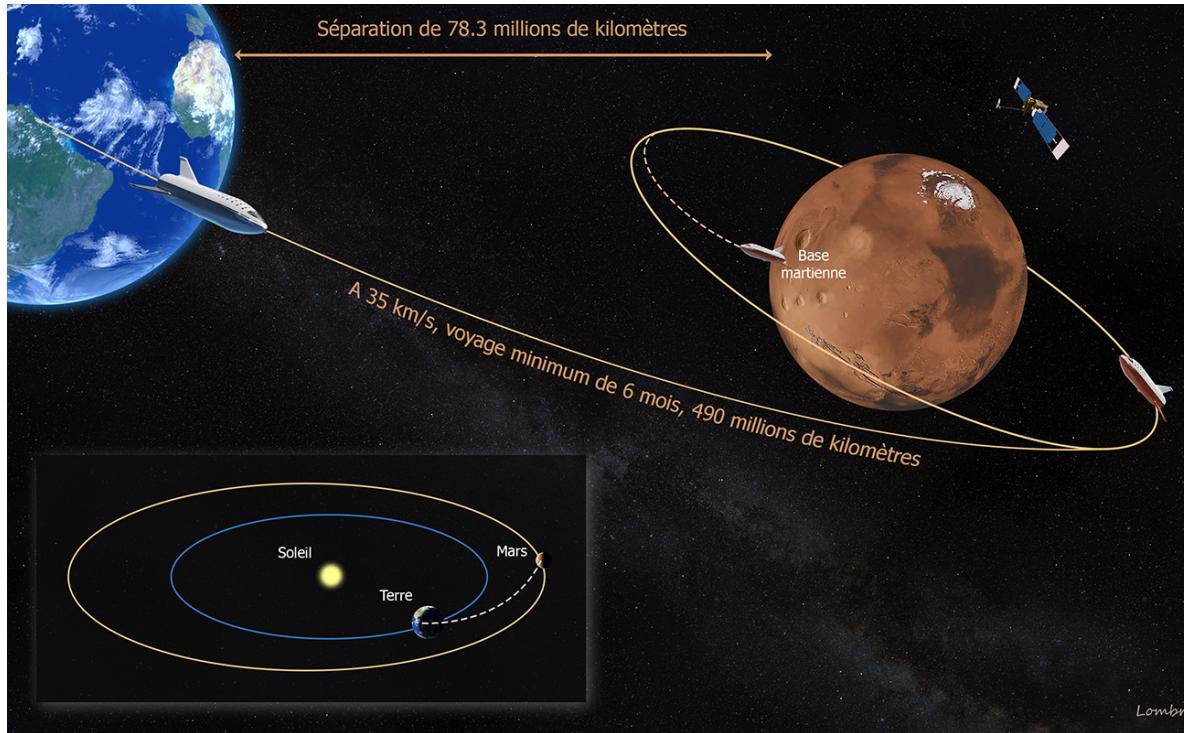


COLONISER MARS

Modélisation de l'envoi d'une sonde sur Mars

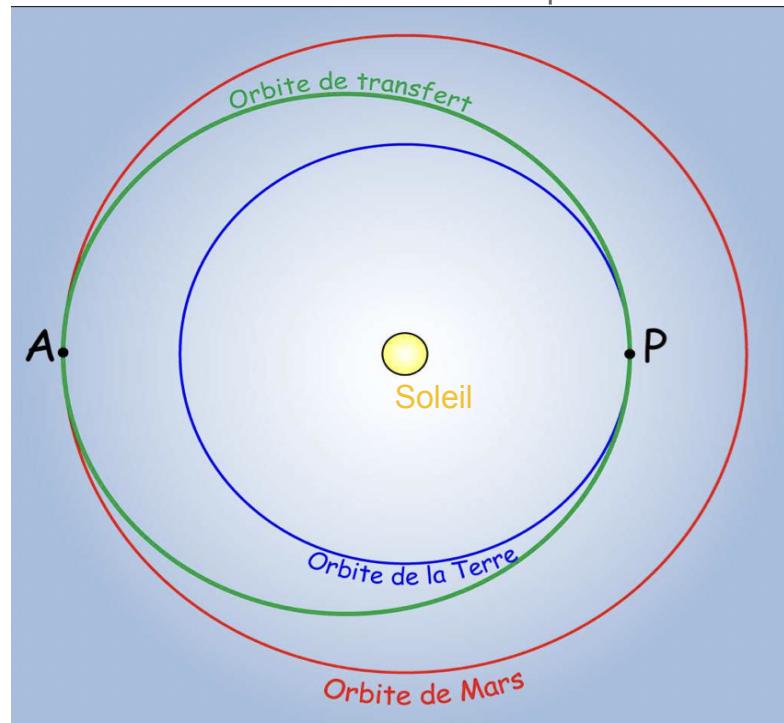
HERBI Hania
LEVY Ilona

Objectif



Première partie : Changement d'orbite

Référentiel héliocentrique



Outils mathématiques et code

- **On commence** par définir la fonction calculant la vitesse de Hohmann définie comme étant la vitesse du changement d'orbite par :

$$V = \sqrt{\frac{2\mu}{r_1} - \frac{2\mu}{r_1 + r_2}}$$

avec $\mu = G * M_{soleil}$, r_1 rayon d'orbite de la Terre et r_2 le rayon d'orbite de Mars.

La fonction prend en entrée les rayons r_1 et r_2 :

```
def vitesse_hohmann(r1, r2):
    """Calcule la vitesse nécessaire pour une transfert de Hohmann entre deux orbites circulaires."""
    mu = G * mSoleil
    v1 = np.sqrt(mu / r1)
    a = (r1 + r2) / 2 # a est le demi grand axe de l'ellipse que va faire le vaisseau en changeant d'orbite.
    v_transfer = np.sqrt(2 * mu / r1 - mu / a)
    return v_transfer - v1
```

et soustrait à la fin la vitesse initiale de la sonde.

Ensuite,

à partir de la troisième loi de Kepler, on trouve l'angle de déphasage optimal entre la Terre et Mars pour le lancement tel que :

$$\frac{T^2}{a^3} = \frac{4 * \pi^2}{G * M}$$

avec

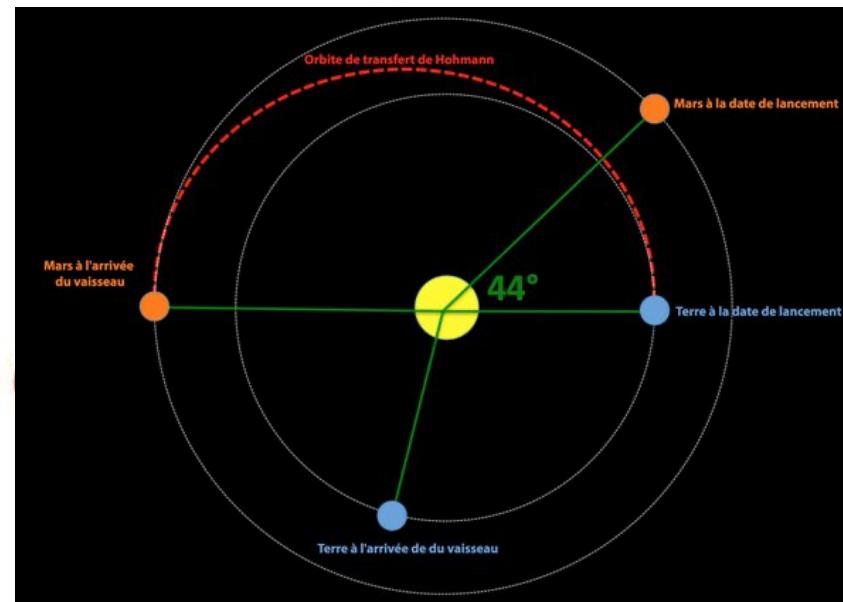
$$a = \frac{r_1 + r_2}{2}$$

Qui donnent:

$$T_{transfert} = \pi * \sqrt{\frac{(r_1 + r_2)^3}{8 * G * M}}$$



$$\theta_{optimale} = \pi - \frac{2\pi}{T_{Mars}} T_{transfert}$$



Ce qui revient à définir la fonction :

```
def angle_phase_optimal():
    """Calcule l'angle de phase optimal pour lancer une mission de la Terre à Mars."""
    T_transfer = np.pi * np.sqrt((distanceTerreSoleil + distanceMarsSoleil)**3 / (8 * G * mSoleil))
    optimal_angle = np.pi - (2 * np.pi / periodeMars) * T_transfer
    return optimal_angle
```

Enfin,

a l'aide des lois de Newton on a l'accélération :

- $a_x = \frac{G * M_s * x}{r^3}$
- $a_y = \frac{G * M_s * y}{r^3}$

et donc la position de la sonde, puisque :

$$\begin{aligned}x &= \int \int a_x(t) dt dt \\y &= \int \int a_y(t) dt dt\end{aligned}$$

Ces intégrations sont résolues numériquement en appliquant la méthode **RK4** :

```
def rk4(t, etat, dt, derivees):
    k1 = dt * derivees(t, etat)
    k2 = dt * derivees(t + 0.5 * dt, etat + 0.5 * k1)
    k3 = dt * derivees(t + 0.5 * dt, etat + 0.5 * k2)
    k4 = dt * derivees(t + dt, etat + k3)
    return etat + (k1 + 2*k2 + 2*k3 + k4) / 6

# Fonction de dérivées spécifique pour le mouvement orbital autour du Soleil
def derivees_heliocentrique(t, etat):
    x, y, vx, vy = etat
    r_carre = x**2 + y**2
    ax = -G * mSoleil * x / r_carre**1.5
    ay = -G * mSoleil * y / r_carre**1.5
    return np.array([vx, vy, ax, ay])
```

Et on simule

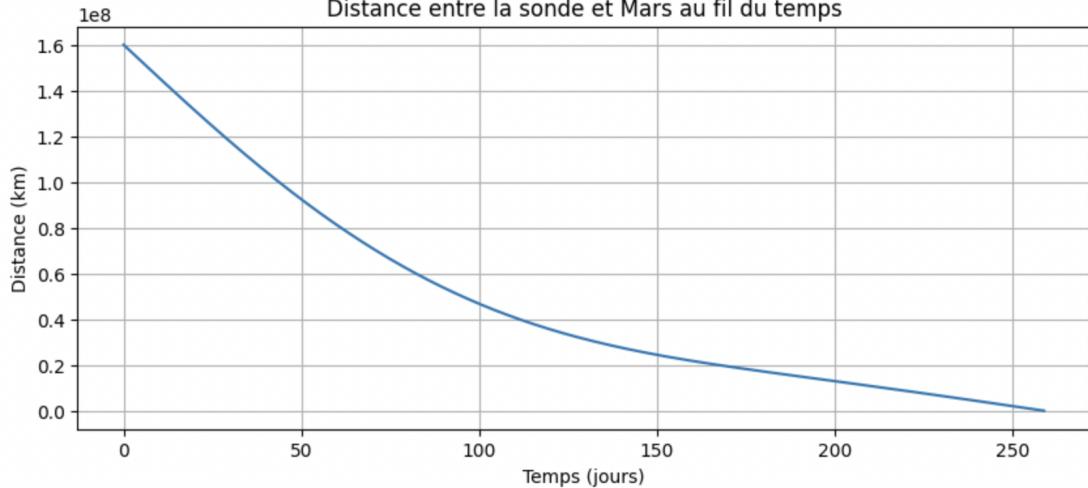
```
def simuler_mission(duree, dt, delta_v, stop_distance=500000):
    num_etapes = int(duree / dt)
    etats = np.zeros((num_etapes, 4)) # x, y, vx, vy
    positions_terre = np.zeros((num_etapes, 2))
    positions_mars = np.zeros((num_etapes, 2))
    distances = np.zeros(num_etapes)
    vitesses = np.zeros(num_etapes)

    angle_phase = angle_phase_optimal()
    vitesse_initiale = delta_v + np.sqrt(G * mSoleil / distanceTerreSoleil)

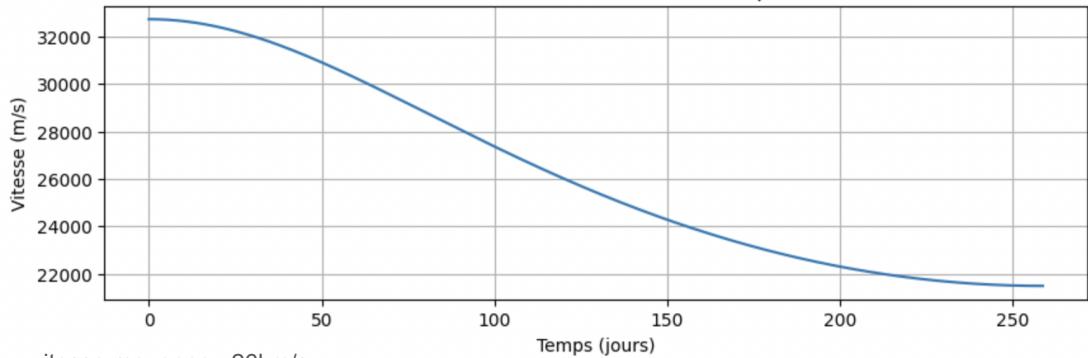
    for i in range(num_etapes):
        t = i * dt
        etats[i] = [distanceTerreSoleil, 0, 0, vitesse_initiale] if i == 0 else rk4(t, etats[i-1], dt, derivees_heliocentrique)
        positions_terre[i] = [distanceTerreSoleil * np.cos(2 * np.pi * t / periodeTerre), distanceTerreSoleil * np.sin(2 * np.pi * t / periodeTerre)]
        positions_mars[i] = [distanceMarsSoleil * np.cos(2 * np.pi * t / periodeMars + angle_phase), distanceMarsSoleil * np.sin(2 * np.pi * t / periodeMars)]
        distances[i] = np.linalg.norm(etats[i, :2] - positions_mars[i]) - rayon_mars
        vitesses[i] = np.linalg.norm(etats[i, 2:4])
        if distances[i] <= stop_distance:
            days, remainder = divmod(i * dt, 86400)
            hours, remainder = divmod(remainder, 3600)
            minutes, seconds = divmod(remainder, 60)
            print(f"La sonde est à 500km de Mars au bout de: {days} jours, {hours} heures, {minutes} minutes, {seconds} secondes")
            num_etapes = i + 1 # Ajuste Le nombre de pas pour arrêter à la distance spécifiée.
            break
```

```
position_finale = simuler_mission(duree, dt, delta_v, stop_distance)
```

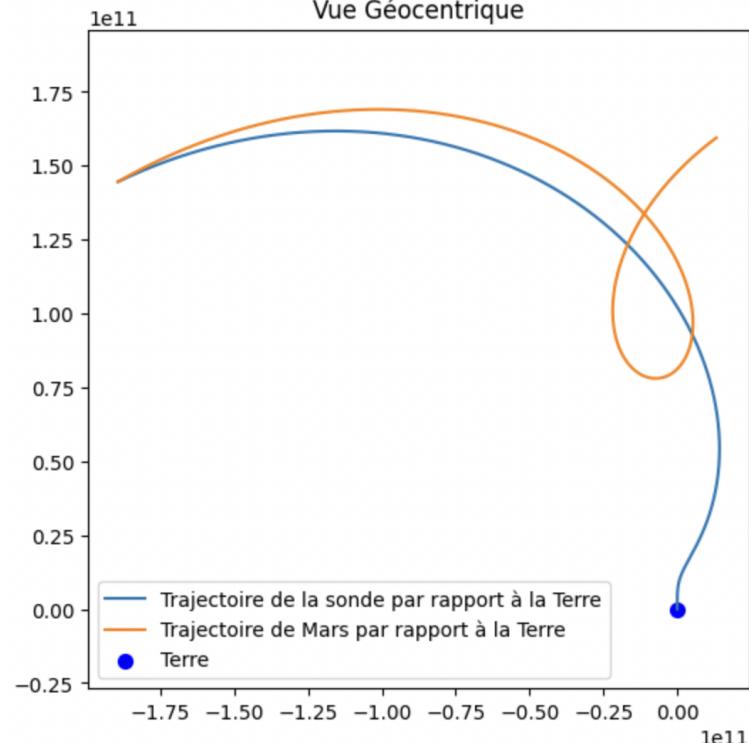
Distance entre la sonde et Mars au fil du temps



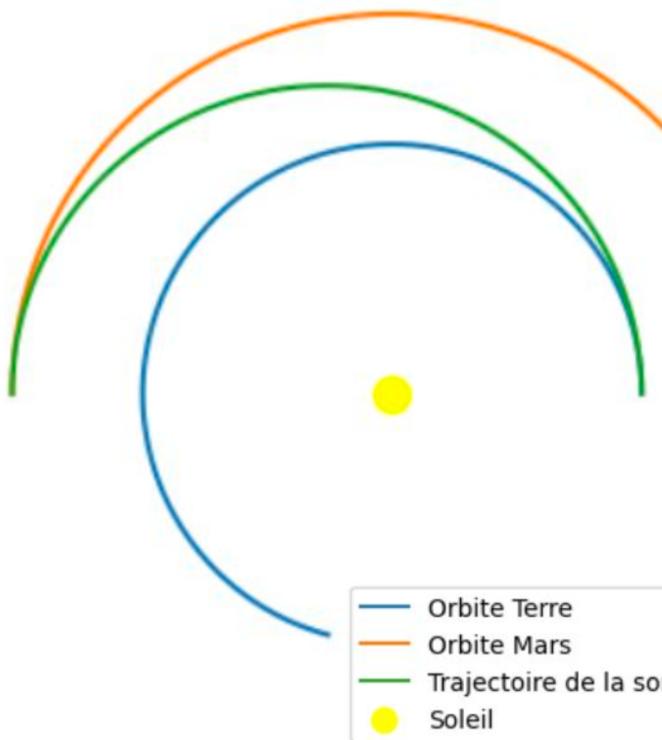
Vitesse de la sonde au fil du temps



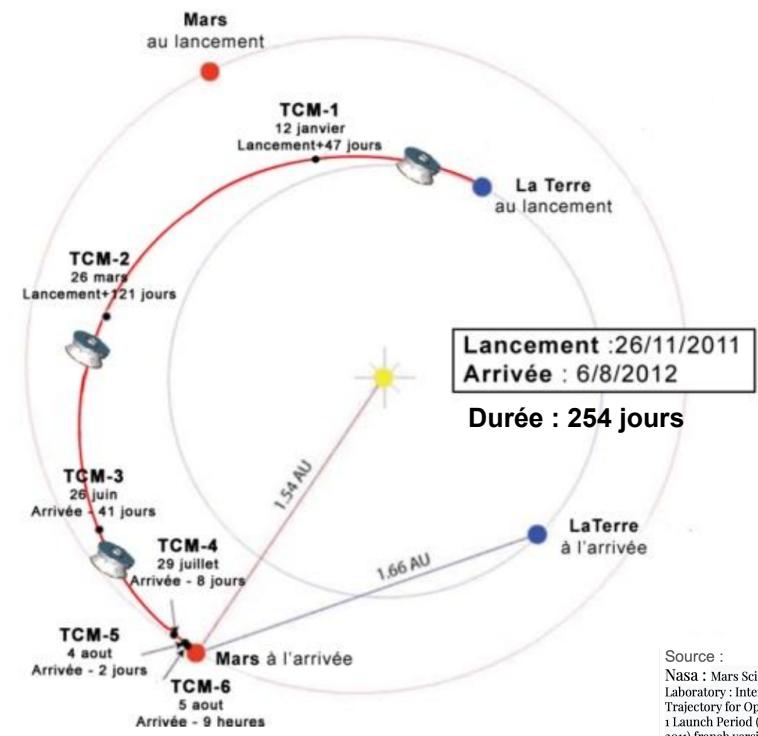
Vue Géocentrique



Vue Héliocentrique



Mission Curiosity:

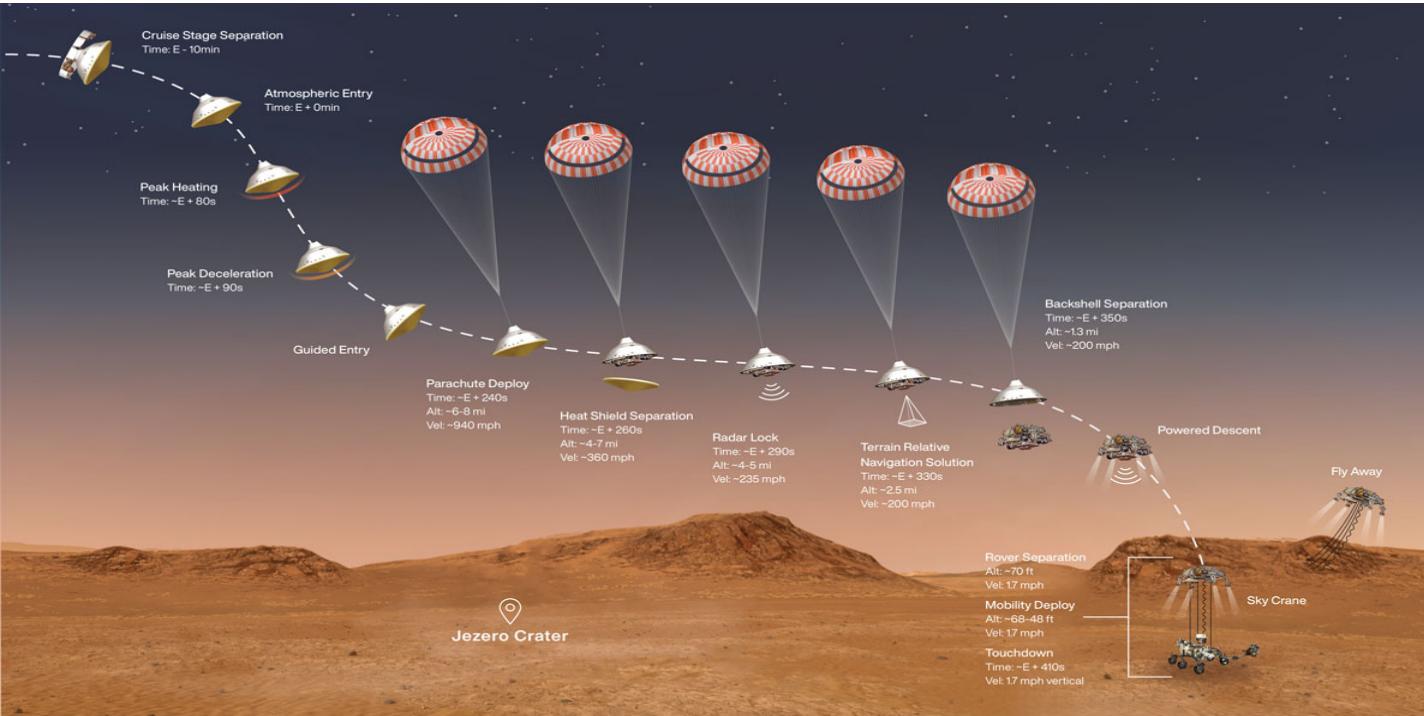


La sonde est à 500km de Mars au bout de: 258 jours, 19 heures

Pour plus de précision

- La trajectoire de Mars est elliptique et sa vitesse n'est pas constante.
- Les effets gravitationnels de la Terre et de Mars sur la sonde, même très faibles devant ceux du Soleil, restent présents et en vigueur.
- Pour le choix du moment du lancement, on n'a que calculé l'angle de déphasage entre les deux planètes alors qu'en réalité il faut aussi prendre en considération la position de la Lune.

Deuxième partie: Amarsissage, atterrissage sur Mars



Référentiel de Mars

Source : Science-Et-Avenir :

Atterrissage de Perseverance sur Mars

Outils mathématiques et code

Dans le référentiel de Mars, et proche de ce dernier, on a :

$$a_x = -G \frac{m_{Mars}}{r^3} x, \quad a_y = -G \frac{m_{Mars}}{r^3} y$$

Pour atterrir il faut un freinage, que l'on ajoute aux accélérations ci-dessus

$$a_x+ = - \left(\frac{\text{force de freinage}}{m_{sonde}} \right) \frac{v_x}{\|\mathbf{v}\|}$$

$$a_y+ = - \left(\frac{\text{force de freinage}}{m_{sonde}} \right) \frac{v_y}{\|\mathbf{v}\|}$$

avec :

$$\text{force de freinage} := f(y) = \begin{cases} 15000N & \text{si } y \leq 150\text{km} \\ 0N & \text{sinon.} \end{cases}$$

```
def derivee_mars(t, etat, d_frein=150000, force=15000, masse_sonde=1000):
```

```
x, y, vx, vy = etat
r_carre = x**2 + y**2
r = np.sqrt(r_carre)      #norme distance du vaisseau à mars
ax = -G * mMars * x / r_carre **1.5
ay = -G * mMars * y / r_carre **1.5
```

```
# Décélération linéaire
```

```
if r - rayon_mars < d_frein:
    v_norm = np.sqrt(vx**2 + vy**2)
    if v_norm != 0: # Éviter la division par zéro
        ax += -(force/masse_sonde) * (vx/v_norm)
        ay += -(force/masse_sonde) * (vy/v_norm)
```

```
return np.array([vx, vy, ax, ay])
```

Utilisation de la fonction derivee_mars et rk4 de la partie 1 pour résoudre les équations.

```

def orbite_mars(position_relative_mars, dt, temps_chute):

    # Convertir la position relative à la surface de Mars en position relative au centre de Mars
    position_initiale = [position_relative_mars[0], position_relative_mars[1] + rayon_mars]

    # Calcul de la vitesse orbitale initiale pour une orbite circulaire
    r_init = np.linalg.norm(position_initiale) #norme position initiale
    v_orbite = np.sqrt(G * mMars / r_init)
    theta = np.arctan2(position_initiale[1], position_initiale[0]) + np.pi/2
    v_init = np.array([np.cos(theta), np.sin(theta)]) * v_orbite

    # Définir l'état initial de la sonde
    etat = np.array([position_initiale[0], position_initiale[1], v_init[0], v_init[1]])
    trajet = [etat.copy()]
    vitesses = [np.linalg.norm(etat[2:])]
    altitudes = [np.linalg.norm(etat[:2]) - rayon_mars]
    t = 0
    times = [t]
    decel_start_time = None

    # Simulation jusqu'à la chute
    while t <= temps_chute:
        etat = rk4(t, etat, dt, derivee_mars)
        trajet.append(etat.copy())
        vitesses.append(np.linalg.norm(etat[2:]))
        altitudes.append(np.linalg.norm(etat[:2]) - rayon_mars)
        times.append(t)
        t += dt

    etat[2:] = 0 # Vitesse nulle pour initier la chute

    # Simulation de la chute sous gravité
    while np.linalg.norm(etat[:2]) > rayon_mars:
        etat = rk4(t, etat, dt, derivee_mars)
        trajet.append(etat.copy())
        vitesses.append(np.linalg.norm(etat[2:]))
        altitudes.append(np.linalg.norm(etat[:2]) - rayon_mars)
        times.append(t)
        t += dt

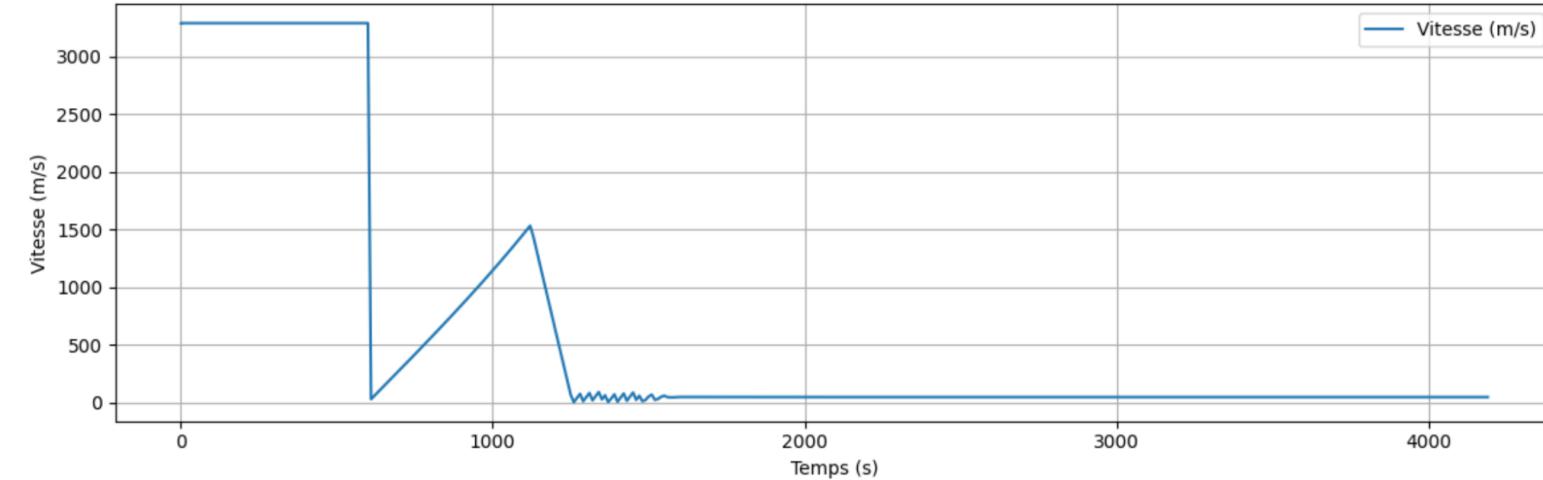
    final_speed = np.linalg.norm(etat[2:]) # Enregistrer vitesse finale

```

Pour la vitesse en orbite:

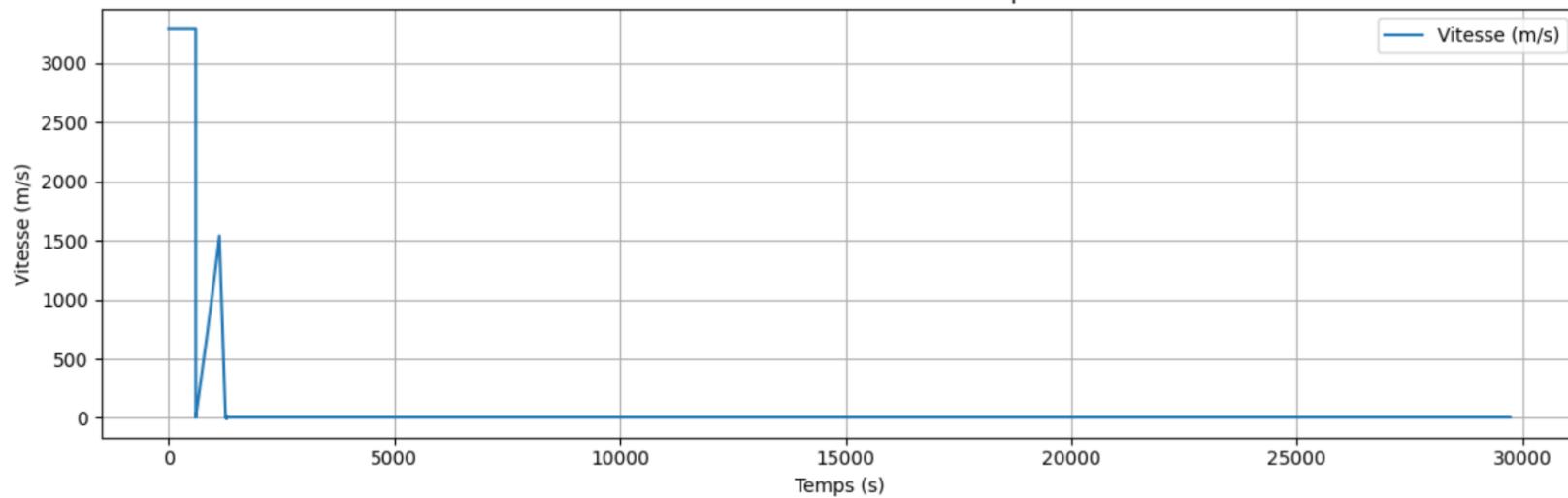
$$v_{\text{orbite}} = \sqrt{\frac{G \cdot m_{\text{Mars}}}{r_{\text{init}}}}$$

Vitesse de la sonde au fil du temps



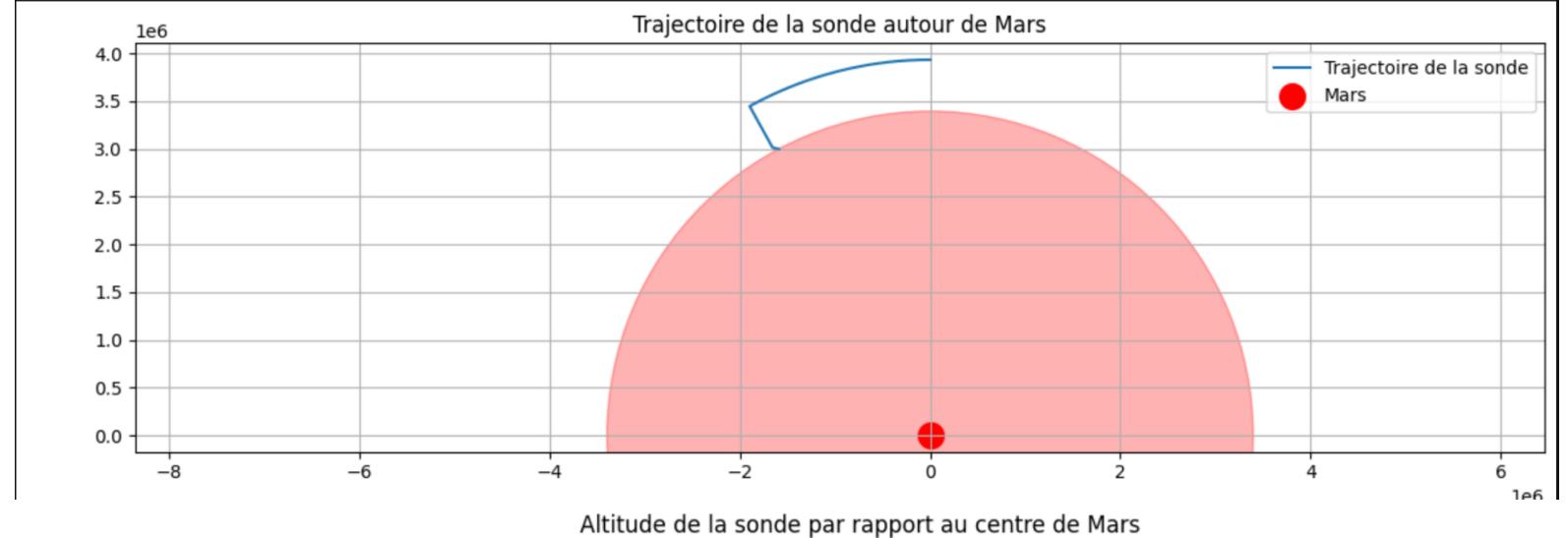
$dt = 10 \text{ sec}$
 $v_{\text{fin}} = 45 \text{ m/s}$

Vitesse de la sonde au fil du temps

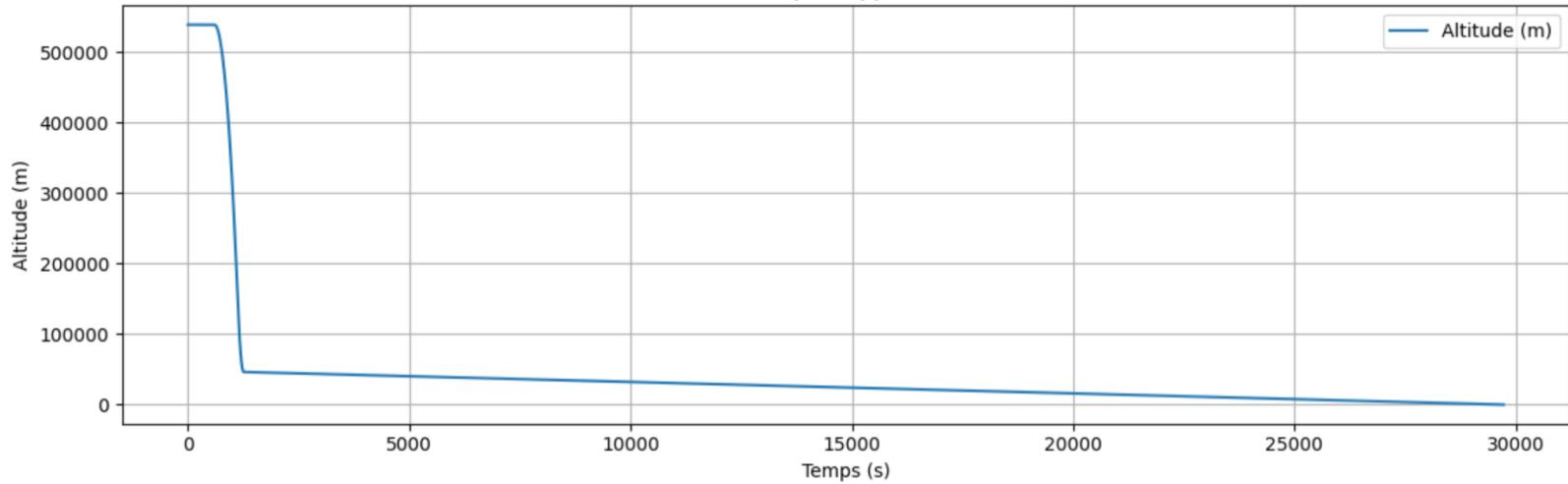


$dt = 1 \text{ sec}$
 $v_{\text{fin}} = 4.5 \text{ m/s}$

La sonde a touché le sol de Mars à 8 heures, 15 minutes, 26 secondes avec une vitesse de 4.55 m/s.



Altitude de la sonde par rapport au centre de Mars



Pour plus de précision

- Ici on a un freinage constant.
- L'intégration et la précision dépendent du pas de la simulation.
- L'initialisation de la chute est trop brutale et simplifiée.
- Mars à une atmosphère et une météo complexe.

Conclusion

**Nous pouvons dire que nous avons atteint l'objectif du projet :
le modèle est confirmé par les vraies données des missions
précédentes et sa compatibilité justifie les hypothèses.**

Mission colonisation-Mars accomplie !

Merci pour votre attention !



pov : Martian looking at the rover