Гравитационное поле Земли

✐**Задание:** вычислить и построить графики зависимости напряженности гравитационного поля Земли от радиус-вектора (начало в центре Земли) + визуализировать векторное поле.

▶ Начнем с **классической теории тяготения**:

Напряженность гравитационного поля - это векторная величина, характеризующая это поле в фиксированной точке и численно равная отношению гравитационной силы F, действующей на неподвижную пробную частицу эталонной массы m_0 в этой точке. \ Тогда выражение для напряженности гравитационного поля выглядит:

$$\Gamma = \frac{F}{m_0}$$

В классической теории тяготения значение гравитационной силы может быть записано в упрощенной форме, если полагать, что источником гравитационного поля является однородное тело сферической формы массой M_3 и радиусом R_3, то есть:

$$\Gamma = \frac{-\frac{Gm_0M_3}{\left(R_3+r\right)^2}}{m_0} = -\frac{GM_3}{\left(R_3+r\right)^2}, \quad r \in \mathrm{R}_+ \cup \{0\}$$

Заметим, что зависимость $\Gamma(R)$ валидна для точек за границей моделируемой сферы Земли. \ Например, потенциал точки за сферой равен $\psi = -\frac{GM_3}{R_3+r}$, тогда:

$$\vec{\Gamma} \underset{-\nabla\psi}{\rightarrow} -\frac{GM_3}{\left(R_3+r\right)^3} \cdot \overrightarrow{\left(R_3+r\right)}$$

Или, что эквивалентно через принцип эквивалентности инерциальной и гравитационной масс:

$$\vec{F} = m_0\vec{g} = -\frac{Gm_0M_3}{\left(R_3+r\right)^3} \cdot \overrightarrow{\left(R_3+r\right)} \Rightarrow \vec{g} \equiv \vec{\Gamma} = -\frac{GM_3}{\left(R_3+r\right)^3} \cdot \overrightarrow{\left(R_3+r\right)}$$

Заметим, что внутри сферы (геоида Земли) потенциал постоянен, то есть напряженность линейно растет до значения $\left[-\frac{GM_3}{R_3^2}\right]$.

\ То есть описание функции зависимости напряженности гравитационного поля от радиус-вектора до фиксированной точки можно дать следующим образом:

$$\Gamma = \begin{cases} -\frac{GM_3}{R_3^3} \cdot r, & r \in \left[0; R_3\right] \\ -\frac{GM_3}{r^2}, & r > R_3 \end{cases}$$

Теперь относительно потенциала гравитационного поля, которое создается нашим однородным шаром - **Землей**:

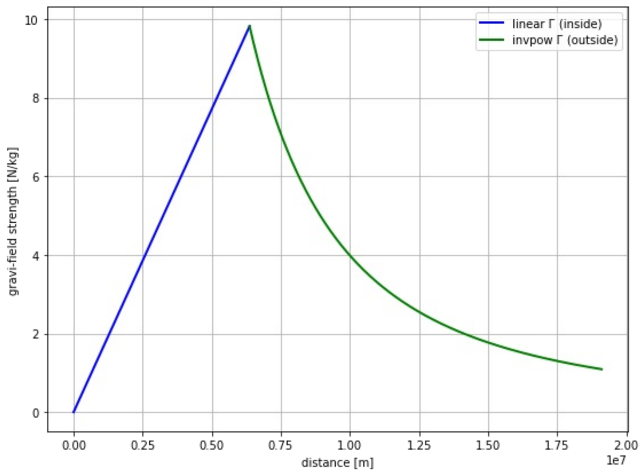
- Снаружи потенциал равен $\psi = -\frac{GM_3}{r}, \ r \in \left(R_3, +\infty\right)$
- Внутри шара необходимо рассматривать шаровой слой переменной массы, тогда для этого случая выражение силы тяготения записывается $|F| = \frac{Gm_0M_{layer}}{r^2} = Gm_0 \cdot \left(\frac{4\rho\pi}{3}\right) \cdot r$, а выражение для потенциала: $\psi = G \cdot \frac{2\pi\rho}{3} \cdot r^2 - 2\pi\rho R_3^2, \ r \in \left[0, R_3\right]$

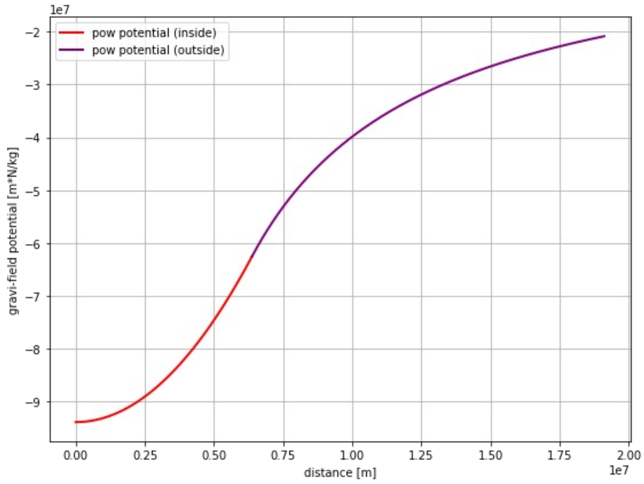
$$\psi = \begin{cases} \frac{GM_3r^2}{2R_3^3} - \frac{3GM_3}{2R_3}, & r \in \left[0; R_3\right] \\ -\frac{GM_3}{r}, & r > R_3 \end{cases}$$

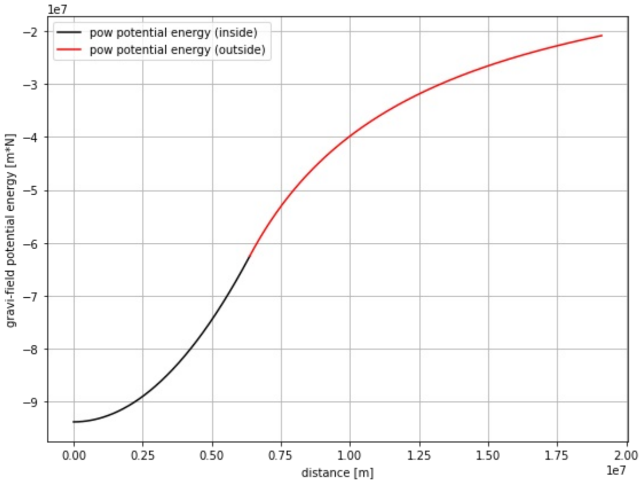
Заметим еще, что потенциальная энергия для единичной массы будет по графику сопадать с потенциалом с точностью до размерностей, из разностей потенциальных энергий понятно, что на поверхности Земли потенциальная энергия равна работе силы тяжести mgh.

In [14]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
import math
```

In [15]:
```
R = 6371 * 10 ** 3
M = 5.9722 * 10 ** 24
G = -6.674 * 10 ** -11
mu = 5515.3
m0 = 1
r1 = np.linspace(0.001, R, 10 ** 5)
r2 = np.linspace(R, 3 * R, 10 ** 5)
plt.figure(figsize=[20, 15])
# strength (Г)
u1 = (-G * M / R ** 3) * r1
u2 = -G * M / r2 ** 2
ax = plt.subplot(2, 2, 1)
ax.plot(r1, u1, color="blue", label="linear Г (inside)", linewidth = 2)
ax.plot(r2, u2, color="green", label="invpow Г (outside)", linewidth = 2)
ax.set_xlabel("distance [m]")
ax.set_ylabel("gravi-field strength [N/kg]")
ax.grid()
ax.legend()
# potential (\psi)
pu1 = - G * M * r1 ** 2 / (2 * R**3) + 3 * G * M / (2 * R)
pu2 = G * M / r2
pax = plt.subplot(2, 2, 2)
pax.plot(r1, pu1, color="red", label="pow potential (inside)", linewidth = 2)
pax.plot(r2, pu2, color="purple", label="pow potential (outside)", linewidth = 2)
pax.set_xlabel("distance [m]")
pax.set_ylabel("gravi-field potential [m*N/kg]")
pax.legend()
pax.grid()
# potential energy
peu1 = (- G * M * r1 ** 2 / (2 * R**3) + 3 * G * M / (2 * R)) * (m0)
peu2 = (G * M / r2) * (m0)
peax = plt.subplot(2, 2, 3)
peax.plot(r1, peu1, color="k", label="pow potential energy (inside)")
peax.plot(r2, peu2, color="red", label="pow potential energy (outside)")
peax.set_xlabel("distance [m]")
peax.set_ylabel("gravi-field potential energy [m*N]")
peax.legend()
peax.grid()
plt.show()
```







In [16]:

```python
class Vector(list):
    def __init__(self, *el):
        for e in el:
            self.append(e)
    def __add__(self, other):
        if type(other) is Vector:
            assert len(self) == len(other), "Error 0"
            r = Vector()
            for i in range(len(self)):
                r.append(self[i] + other[i])
            return r
        else:
            other = Vector.emptyvec(lens=len(self), n=other)
            return self + other
    def __sub__(self, other):
        if type(other) is Vector:
            assert len(self) == len(other), "Error 0"
            r = Vector()
            for i in range(len(self)):
                r.append(self[i] - other[i])
            return r
        else:
            other = Vector.emptyvec(lens=len(self), n=other)
            return self - other
    def __mul__(self, other):
        if type(other) is Vector:
            assert len(self) == len(other), "Error 0"
            r = Vector()
            for i in range(len(self)):
                r.append(self[i] * other[i])
            return r
        else:
            other = Vector.emptyvec(lens=len(self), n=other)
            return self * other
    def __truediv__(self, other):
        if type(other) is Vector:
            assert len(self) == len(other), "Error 0"
            r = Vector()
            for i in range(len(self)):
                r.append(self[i] / other[i])
            return r
        else:
            other = Vector.emptyvec(lens=len(self), n=other)
            return self / other
    def __pow__(self, other):
        if type(other) is Vector:
            assert len(self) == len(other), "Error 0"
```

```
            r = Vector()
            for i in range(len(self)):
                r.append(self[i] ** other[i])
            return r
        else:
            other = Vector.emptyvec(lens=len(self), n=other)
            return self ** other
    def __mod__(self, other):
        return sum((self - other) ** 2) ** 0.5
    def mod(self):
        return self % Vector.emptyvec(len(self))
    def dim(self):
        return len(self)
    def __str__(self):
        if len(self) == 0:
            return "Empty"
        r = [str(i) for i in self]
        return "< " + " ".join(r) + " >"
    def _ipython_display_(self):
        print(str(self))
    @staticmethod
    def emptyvec(lens=2, n=0):
        return Vector(*[n for i in range(lens)])
    @staticmethod
    def randvec(dim):
        return Vector(*[random.random() for i in range(dim)])
    @staticmethod
    def centervec(dim):
        return Vector(*[0.5 for i in range(dim)])
```

In [17]:

```
class Point:
    def __init__(self, coords, mass=1.0, q=1.0, speed=None, **properties):
        self.coords = coords
        if speed is None:
            self.speed = Vector(*[0 for i in range(len(coords))])
        else:
            self.speed = speed
        self.acc = Vector(*[0 for i in range(len(coords))])
        self.mass = mass
        self.__params__ = ["coords", "speed", "acc", "q"] + list(properties.keys())
        self.q = q
        for prop in properties:
            setattr(self, prop, properties[prop])
    def move(self, dt):
        self.coords = self.coords + self.speed * dt
    def accelerate(self, dt):
        self.speed = self.speed + self.acc * dt
    def accinc(self, force):
        self.acc = self.acc + force / self.mass
    def clean_acc(self):
        self.acc = self.acc * 0
    def __str__(self):
        r = ["Point {"]
        for p in self.__params__:
            r.append("  " + p + " = " + str(getattr(self, p)))
        r += ["}"]
        return "\n".join(r)
    def _ipython_display_(self):
        print(str(self))
```

In [18]:

```
class InteractionField:
    def __init__(self, F):
        self.points = []
        self.F = F
    def move_all(self, dt):
        for p in self.points:
            p.move(dt)
    def intensity(self, coord):
        proj = Vector(*[0 for i in range(coord.dim())])
        single_point = Point(Vector(), mass=1.0, q=1.0)
        for p in self.points:
            if coord % p.coords < 10 ** (-10):
                continue
            d = p.coords % coord
            fmod = self.F(single_point, p, d) * (-1)
            proj = proj + (coord - p.coords) / d * fmod
        return proj
    def step(self, dt):
        self.clean_acc()
        for p in self.points:
            p.accinc(self.intensity(p.coords) * p.q)
            p.accelerate(dt)
            p.move(dt)
    def clean_acc(self):
        for p in self.points:
            p.clean_acc()
    def append(self, *args, **kwargs):
        self.points.append(Point(*args, **kwargs))
```

```
    def gather_coords(self):
        return [p.coords for p in self.points]
```

In [19]:

```
def sigm(x):
    return 0.9 / (1 + 1.10 ** (-x/1000))
# Визуализация векторного поля
if True:
    u = InteractionField(lambda p1, p2, r: 300000 * -p1.q * p2.q / (r ** 2 + 0.1))
    u.append(Vector.centervec(2) * 10, q=random.random() - 0.5)
    fig = plt.figure(figsize=[15, 15])
    res = []
    STEP = 0.4
    for x in np.arange(0, 10, STEP):
        for y in np.arange(0, 10, 0.9*STEP):
            inten = u.intensity(Vector(x, y))
            F = inten.mod()
            inten /= inten.mod() * 3
            res.append(([x - inten[0] / 2, x + inten[0] / 2], [y - inten[1] / 2, y + inten[1] / 2], F))
    for r in res:
        plt.plot(r[0], r[1], color=(sigm(r[2]), 0.4, 0.8 * (1 - sigm(r[2]))), linewidth = 3)
    plt.show()
```