Programming assignment #1: rootfinding

Problem 1. Write a Python function:

```
roots = findroots(p, a, b)
```

whose arguments are:

- `p`: a list or `ndarray` of double-precision floating point numbers of length $n+1$ defining a degree n polynomial such that `p[i]` contains the value of p_i so that `p` defines the polynomial $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots p_n x^n$,
- and `a` and `b`: two *finite* double-precision floating point numbers which together define an interval $[a, b]$,

and which computes all real roots of $p(x)$ on the interval $[a, b]$. The function should return a list of the real roots *in increasing order*: if p has k roots x_i such that $a \leq x_1 \leq x_2 \leq \cdots \leq x_k \leq b$, then `roots[i]` (`1 <= i <= k`) gives the value of x_i. If there are no roots, then `f` returns an empty list (i.e. `len(roots) == 0`).

Problem 2. An algebraic surface (click through to see pictures of many examples) is defined as the locus of points which satisfies:

$$p(x, y, z) = 0, \qquad (x, y, z) \in \mathbb{R}^3, \tag{1}$$

where p is a multivariable polynomial. Goursat's surface is a quartic algebraic surface defined by (1) where:

$$p(x, y, z) = x^4 + y^4 + z^4 + a(x^2 + y^2 + z^2)^2 + b(x^2 + y^2 + z^2) + c = 0, \tag{2}$$

for some choice of the parameters $a, b, c \in \mathbb{R}$.

Using `findroots`, we will use raytracing to render an image of Goursat's surface. We pick a point (x_0, y_0, z_0), a unit ray direction $\boldsymbol{d} = (d_x, d_y, d_z)$, and define the *ray*:

$$\boldsymbol{r}(t) = (x_0 + td_x, y_0 + td_y, z_0 + td_z), \qquad t \geq 0. \tag{3}$$

We then find the values of t for which (1) holds:

$$p(\boldsymbol{r}(t)) = 0, \qquad t \geq 0. \tag{4}$$

Note that the composition of a multivariate polynomial with a single variable polynomial is again a polynomial. This means that we can use `findroots` to solve (4).

In our simplified raytracing, we will set up a grid of rays, one for each pixel in an image, solve (4) using `findroots` means, and assign a color to that pixel based on the result. To color each pixel, we will use a simple Lambertian model of reflectance:

- We will represent colors as 3-tuples of floating-point values, (r, g, b), where $r, g, b \in [0, 1]$ are the red, green, and blue values in the RGB color model.
- If we let C be the color of our surface, we will additionally shade it based on the angle that the ray makes with the surface. If $\boldsymbol{n}(x, y, z)$ is a unit surface normal, then for each ray which intersects the surface, we let $\cos(\alpha_{ij}) = -\boldsymbol{n} \cdot \boldsymbol{d}$, and set the corresponding pixel value to:
$$C_{ij} = \cos(\alpha_{ij})C. \tag{5}$$

Some comments:

- To use `findroots` to solve (4), we need to write $p(\boldsymbol{r}(t))$ as a polynomial in t. This is tricky to do automatically using numpy, but you are welcome to try. Two other options: use `sympy`, or write down the polynomial by hand and then implement it as a new Python function (e.g., `p_of_r(t, r, d, a, b, c)`—note the dependence on the parameters).