Runge–Kutta methods

The midpoint method can be written as:

$$\begin{aligned} x_{n+1} &= x_n + h \\ y_{n+1} &= y_n + K_1 h \\ K_0 &= f(x_n, y_n) \\ K_1 &= f\left(x_n + \frac{h}{2}, y_n + K_0 \frac{h}{2}\right) \end{aligned}$$

Note that replacing the rule by $y_{n+1} = y_n + K_0 h$ results in Euler's method. Indeed, both K_0 and K_1 are approximations of the slope y' that we need for stepping from x_n to $x_{n+1} = x_n + h$.

Adding further such approximations K_i to the mix, one can eliminate further terms in the error expansion and obtain higher order methods known as **Runge–Kutta methods**.

The midpoint method is an example of a Runge–Kutta method of order 2 (but there are others as well).

https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

Of particular practical importance is the following instance:

(Runge–Kutta method of order 4)

$$\begin{aligned} x_{n+1} &= x_n + h \\ y_{n+1} &= y_n + \frac{1}{6}(K_0 + 2K_1 + 2K_2 + K_3)h \\ K_0 &= f(x_n, y_n) \\ K_1 &= f\left(x_n + \frac{h}{2}, y_n + K_0 \frac{h}{2}\right) \\ K_2 &= f\left(x_n + \frac{h}{2}, y_n + K_1 \frac{h}{2}\right) \\ K_3 &= f(x_n + h, y_n + K_2 h) \end{aligned}$$

Comment. Note how each of K_0, K_1, K_2, K_3 is an approximation of y' on the interval $[x_n, x_{n+1}]$ (with K_0 approximating $y'(x_n)$ and K_3 approximating $y'(x_{n+1})$). By taking the appropriate weighted average, we are able to get an approximation with a higher order.

Advanced comment. Note that the weights (with K_1 and K_2 combined because they both correspond to the midpoint $x_n + h/2$) are the same as in Simpson's rule for numerical integration. That is more than a coincidence. Indeed, if $f(x, y) = f(x)$ does not depend on y, then solving the DE is equivalent to integrating $f(x)$ and the Runge–Kutta method of order 4 turns into Simpson's rule.

Example 147. Python Let us implement the Runge–Kutta method of order 4.

```
>>> def runge_kutta4(f, x0, y0, xmax, n):
        h = (xmax - x0) / n
        ypoints = [y0]
        for i in range(n):
            K0 = f(x0,y0)
            K1 = f(x0+h/2, y0+K0*h/2)
            K2 = f(x0+h/2, y0+K1*h/2)
            K3 = f(x0+h, y0+K2*h)
            y0 = y0 + (K0 + 2*K1 + 2*K2 + K3)*h/6
            x0 = x0 + h
            ypoints.append(y0)
        return ypoints
```

First, for comparison with earlier methods, let us apply the method to the IVP $y'=y$, $y(0)=1$, which has the exact solution $y(x)=e^x$ with $y(1)=e\approx 2.718$.

```
>>> def f_y(x, y):
        return y

>>> runge_kutta4(f_y, 0, 1, 1, 4)

   [1, 1.2840169270833333, 1.648699469036526, 2.1169580259162033, 2.718209939201323]
```

The following convincingly illustrates that the error is indeed $O(h^4)$.

```
>>> from math import e

>>> [runge_kutta4(f_y, 0, 1, 1, 10**n)[-1] - e for n in range(6)]

   [-0.009948495125712054, -2.0843238792700447e-06, -2.2464119453502462e-10, -
   2.042810365310288e-14, 1.1546319456101628e-14, 6.217248937900877e-15]
```

Pause for a moment to really appreciate how much better these errors are in comparison with Euler's method! Whereas computing 10^5 values with Euler's method resulted in an error of $1.36\cdot 10^{-5}$, we are now able to obtain an error of $2.04\cdot 10^{-14}$ with only 10^3 values.

As a second example, let us consider as in Example 144 the IVP $y'=\cos(x)y$ with $y(0)=1$, which has the exact solution $y(x)=e^{\sin(x)}$ with $y(2)=e^{\sin(2)}\approx 2.48258$.

```
>>> def f_cosx_y(x, y):
        return cos(x)*y

>>> runge_kutta4(f_cosx_y, 0, 1, 2, 4)

   [1, 1.614859377441316, 2.3191895982789603, 2.7107641474177457, 2.481902218021582]
```

The following again convincingly illustrates that the error is indeed $O(h^4)$.

```
>>> from math import e

>>> [runge_kutta4(f_cosx_y, 0, 1, 2, 10**n)[-1] - e**sin(2) for n in range(5)]

   [-0.12999578105593113, -1.726387102785054e-05, -1.6494263732624859e-09, -
   1.6431300764452317e-13, 3.419486915845482e-13]
```

Important comment. Note that, in contrast to Example 144, we did not have to compute partial derivatives of $f(x,y)=\cos(x)y$ by hand. Instead, we were able to simply use $\cos(x)y$ in our `runge_kutta4` function.

A glance at discretizing PDEs

One of the most important partial differential equations is the following Laplace equation which, for instance, models the steady-state temperature $u(x,y)$ of a region in two-dimensional space.

(Laplace equation)

$$u_{xx}+u_{yy}=0$$

Comment. Here, for instance, $u_{xx}=\frac{\partial^2}{\partial x^2}u(x,y)$ is used to denote two partial derivatives with respect to x.

Comment. The Laplace equation is so important that its solutions have their own name: **harmonic functions**.

Comment. Also known as the "potential equation"; satisfied by electric/gravitational potential functions.

Recall from Calculus III (if you have taken that class) that the gradient of a scalar function $f(x,y)$ is the vector field $\boldsymbol{F}=\operatorname{grad} f=\nabla f=\begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix}$. One says that $\boldsymbol{F}$ is a **gradient field** and f is a **potential function** for $\boldsymbol{F}$ (for instance, $\boldsymbol{F}$ could be a gravitational field with gravitational potential f).

The divergence of a vector field $\boldsymbol{G}=\begin{bmatrix} g(x,y) \\ h(x,y) \end{bmatrix}$ is $\operatorname{div}\boldsymbol{G}=g_x+h_y$. One also writes $\operatorname{div}\boldsymbol{G}=\nabla\cdot\boldsymbol{G}$.

The gradient field of a scalar function f is divergence-free if and only if f satisfies the Laplace equation $\Delta f=0$.

Other notations. $\Delta f=\operatorname{div}\operatorname{grad} f=\nabla\cdot\nabla f=\nabla^2 f$

Boundary conditions. For steady-state temperatures profiles, it is natural to prescribe the temperature on the boundary of a region $R\subseteq\mathbb{R}^2$ (or $R\subseteq\mathbb{R}^3$ in the 3D case).

Comment. Gravitational and electrostatic potentials (not in the vacuum) satisfy the **Poisson equation** $u_{xx}+u_{yy}=f(x,y)$, the inhomogeneous version of the Laplace equation.

One way to describe a unique solution to the Laplace equation is by specifying the values of $u(x,y)$ along the boundary of a region. This is called a Dirichlet problem:

(Dirichlet problem)

$u_{xx}+u_{yy}=0$ within region R

$u(x,y)=f(x,y)$ on boundary of R

In general. A Dirichlet problem consists of a PDE, that needs to hold within a region R, and prescribed values on the boundary of that region ("Dirichlet boundary conditions").

Discretizing the Laplace operator

Recall from Example 115 that the following central difference is an order 2 approximation of $f''(x)$.

$$f''(x)\approx\frac{1}{h^2}[f(x+h)-2f(x)+f(x-h)].$$

Example 148. (discretizing Δ) Use the above central difference approximation for second derivatives to derive a finite difference for $\Delta u = u_{xx} + u_{yy}$ in 2D.

Solution. $\Delta u \approx \frac{1}{h^2}[u(x+h,y)+u(x-h,y)+u(x,y+h)+u(x,y-h)-4u(x,y)]$

Notation. This finite difference is typically represented as $\frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$, the **five-point stencil**.

Comment. Recall that solutions to $\Delta u = 0$ are supposed to describe steady-state temperature distributions. We can see from our discretization that this is reasonable. Namely, $\Delta u = 0$ becomes approximately equivalent to

$$u(x,y) = \frac{1}{4}(u(x+h,y)+u(x-h,y)+u(x,y+h)+u(x,y-h)).$$

In other words, the temperature $u(x,y)$ at a point (x,y) should be the average of the temperatures of its four "neighbors" $u(x+h,y)$ (right), $u(x-h,y)$ (left), $u(x,y+h)$ (top), $u(x,y-h)$ (bottom).

Comment. Think about how to use this finite difference to numerically solve the corresponding Dirichlet problem by discretizing (one equation per lattice point).