1 Periodic Random Vector Fields

Kim et al. [3] demonstrated how Wavelet Noise [1] is used to augment a fluid flow with more turbulence. Algorithm 1 contains our adaption of their multiband sampling algorithm. In our work, the flow is periodic and is sampled from a noise field by using quartic B-spline interpolation.

At point \((x, y)\) the function \(\text{sum\_octaves}\) sums the co-gradient of the wavelet noise field \(data\) with resolution \(n \times n\) from band \(b_{\text{min}}\) to \(b_{\text{max}}\). We refer to the appendix of Cook and DeRose [1] for the computation of the wavelet noise field \(data\), which we synthesize for each band.

2 Visualization of Error and FTLE for Varying Integration Durations

The upsampling of flow maps becomes more difficult the longer the flow maps are traced. In the following, we provide more results for varying integration durations and we deliver more quantitative evaluations for MSE and PSNR for the \(4 \times \) upsampling methods. Fig. 1 and Fig. 2 visualize the error residuals and the FTLE values for \(2 \times \) and \(4 \times \), respectively. As in the main paper, we apply cubic upsampling, SRCNN [2] and ESPCN [4]. We can see that ESPCN at \(2 \times \) upsampling performed very well, whereas artifacts occur in laminar parts of the domain for \(4 \times \) upsampling.

REFERENCES


Algorithm 1: Compute periodic stream function by summing wavelet noise bands that were sampled with quartic B-splines.

```
function quartic_b_spline(double t, double *w):
  w[0] ← t^4/24;
  w[1] ← (−4t^4 + 4t^2 + 6t + 1)/24;
  w[2] ← (6t^5 − 12t^3 + 6t^2 + 4t + 1)/24;
  w[3] ← (−4t^4 + 12t^3 + 6t^2 + 11)/24;
  w[4] ← (1 − t)^4/24

function quartic_b_spline_dt(double t, double *w):
  w[0] ← t^3/6;
  w[1] ← (−4t^4 + 3t^2 + 3t + 1)/6;
  w[2] ← (6t^5 − 9t^3 + 3t^2 + 3t + 1)/6;
  w[3] ← (−4t^4 + 9t^3 − 3t^2 + 3t + 1)/6;
  w[4] ← (−1 − t)^3/24

function eval(int i, int j, double *w0, double *w1, int k, int n, double *data):
  result ← 0
  for j ← −2 to 2 do
    for i ← −2 to 2 do
      c_i ← mod(i + j, k)
      result ← result + w0[i + 2] * w1[j + 2] * data[j + n * c_i]
  return result

function dw_dx(double x, double y, int k, int n, double *data):
  i ← ceil(x − 0.5), i ← ceil(y − 0.5)
  double w0[5], w1[5]
  quartic_b_spline_dt(i, x, 0.5, w0)
  quartic_b_spline(i, y, 0.5, w1)
  return eval(i, i, w0, w1, k, n, data)

function dw_dy(double x, double y, int k, int n, double *data):
  i ← ceil(x − 0.5), i ← ceil(y − 0.5)
  double w0[5], w1[5]
  quartic_b_spline_dt(i, x, 0.5, w0)
  quartic_b_spline(i, y, 0.5, w1)
  return eval(i, i, w0, w1, k, n, data)

function sum_octaves(double x, double y, int b_min, int b_max, int n, double **data):
  vel ← (0.0)
  for b ← b_min to b_max do
    k ← pow(2, b)
    vel ← vel + (−dw_dx(x, y, k, n, data[b]) * 2 * pow(2, −b * b_min)) 
    dw_dx(x, y, k, n, data[b]) * 2 * pow(2, −b * b_min))
  return vel
```
Fig. 1: Error maps and FTLE comparisons in our simulated flow #2000 for different integration durations $\tau$. Here, for a $2 \times$ upsampling.
Fig. 2: Error maps and FTLE comparisons in our simulated flow #2000 for different integration durations $\tau$. Here, for a $4 \times$ upsampling.