Neutrino oscillation probability from a Jpp perspective

M. de Jong

Neutrino oscillation probability

$$P(\nu_1 \to \nu_2) \equiv f(\sin^2\theta_{12}, \Delta m_{21}, \sin^2\theta_{13}, \Delta m_{31}, \sin^2\theta_{23}, \delta_{CP}; E_\nu, \cos\theta)$$

Is it possible to?

A. interpolate $f(\sin^2\theta_{12}, \Delta m_{21}, \sin^2\theta_{13}, \Delta m_{31}, \sin^2\theta_{23}, \delta_{CP}, E_\nu, \cos\theta)$

B. interpolate $f(\sin^2\theta_{12}, \Delta m_{21}, \sin^2\theta_{13}, \Delta m_{31}, \sin^2\theta_{23}, \delta_{CP})$ and interpolate *a posteriori* between $(E_\nu, \cos\theta)$

Part I

8D-table – containing double

```
typedef JMAPLIST< JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap>::maplist          JMaplist_t;

JMultiMap<double, double, JMaplist_t>     zmap;
```

2D-interpolator – returning double

```
typedef   double                                                          data_type;
typedef   JElement2D<double, data_type>                                   element_type;
typedef   JPolintFunction1D<0, element_type, JGridCollection, data_type>  JFunction1D_t

typedef   JMAPLIST<JPolint0FunctionalGridMap>::maplist                    JMaplist1D_t;
typedef   JMultiFunction<JFunction1D_t, JMaplist1D_t>                     JFunction2D_t;
```

6D-interpolator – template

```
typedef   JMAPLIST< JPolint0FunctionalGridMap,                  // sin^2 θ_12
                    JPolint0FunctionalGridMap,                  // Δm_21
                    JPolint1FunctionalGridMap,
                    JPolint1FunctionalGridMap,
                    JPolint1FunctionalGridMap,
                    JPolint1FunctionalGridMap>::maplist         JMaplist6D_t;
```

8D-interpolator – returning double

```
typedef    JMultiFunction<JFunction2D_t, JMaplist6D_t>                    JFunction8D_t;
```

6D-interpolator – returning 2D-table

```
typedef   JCollection<element_type>                          JCollection1D_t;
typedef   JMap<double, JCollection1D_t>                      JCollection2D_t;
typedef   JConstantFunction1D<double, JCollection2D_t>       JConstantFunction1D_t;

typedef   JMultiFunction<JConstantFunctio1D_t, JMaplist6D_t> JFunction6D_t;
```

CPU time

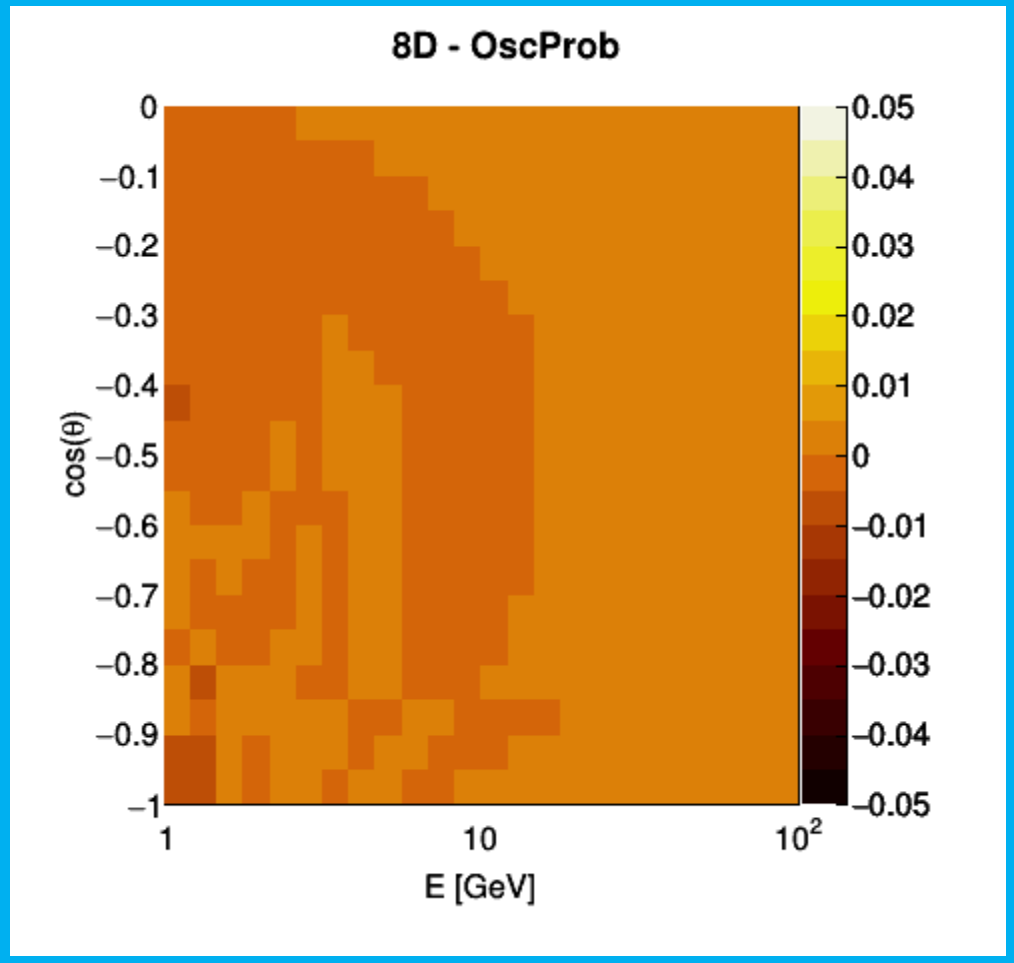
OscProb
Get oscillation probability... OK
9000 μs elapsed

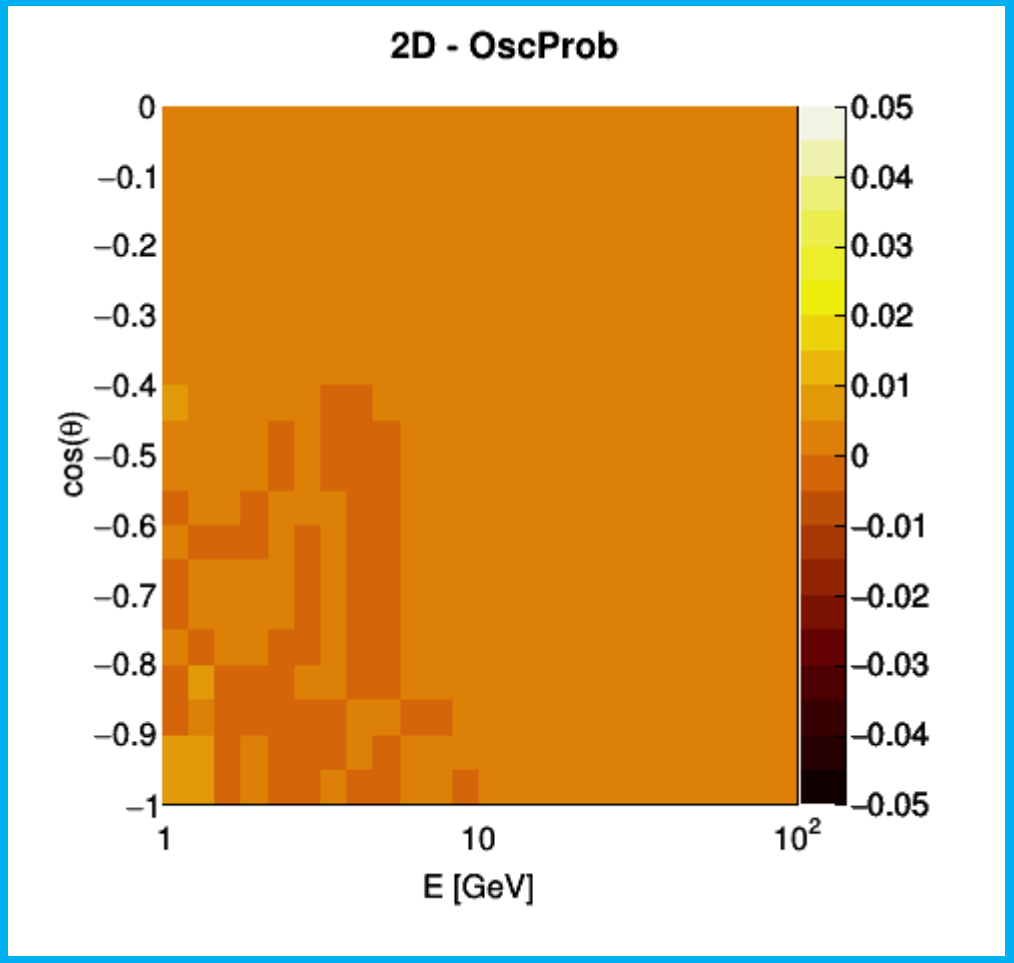
Jpp
Get 8D-interpolation... OK
450 μs elapsed

Jpp
Get 2D-function... OK
375 μs elapsed

Jpp
Get 2D-interpolation... OK
30 μs elapsed

Random point in neutrino parameter space





Part II

- Removing dependence on $(\sin^2\theta_{12}, \Delta m_{21})$
- Combining 10 oscillation channels (ν and $\bar{\nu}$)

$\nu_e \to \nu_e \quad \nu_e \to \nu_\mu \quad \nu_e \to \nu_\tau$

$\nu_\mu \to \nu_\mu \quad \nu_\mu \to \nu_\tau$

6D-table – containing array

```
typedef JArray<10, double>                                  data_type;

typedef JMAPLIST< JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap,
                  JMap>::maplist                            JMaplist_t;

JMultiMap<double, data_type, JMaplist_t>                    zmap;
```

2D-interpolator – returning array

```
typedef   JArray<10, double>                                                  data_type;
typedef   JElement2D<double, data_type>                                       element_type;
typedef   JPolintFunction1D<0, element_type, JGridCollection, data_type>      JFunction1D_t

typedef   JMAPLIST<JPolint0FunctionalGridMap>::maplist                        JMaplist1D_t;
typedef   JMultiFunction<JFunction1D_t, JMaplist1D_t>                         JFunction2D_t;
```

4D-interpolator – template

```
typedef    JMAPLIST< JPolint1FunctionalGridMap,
                     JPolint1FunctionalGridMap,
                     JPolint1FunctionalGridMap,
                     JPolint1FunctionalGridMap>::maplist          JMaplist4D_t;
```

6D-interpolator – returning array

```
typedef    JMultiFunction<JFunction2D_t, JMaplist4D_t>                    JFunction6D_t;
```

4D-interpolator – returning 2D-table

```
typedef   JCollection<element_type>                          JCollection1D_t;
typedef   JMap<double, JCollection1D_t>                      JCollection2D_t;
typedef   JConstantFunction1D<double, JCollection2D_t>       JConstantFunction1D_t;

typedef   JMultiFunction<JConstantFunction2D_t, JMaplist6D_t>   JFunction4D_t;
```

CPU time

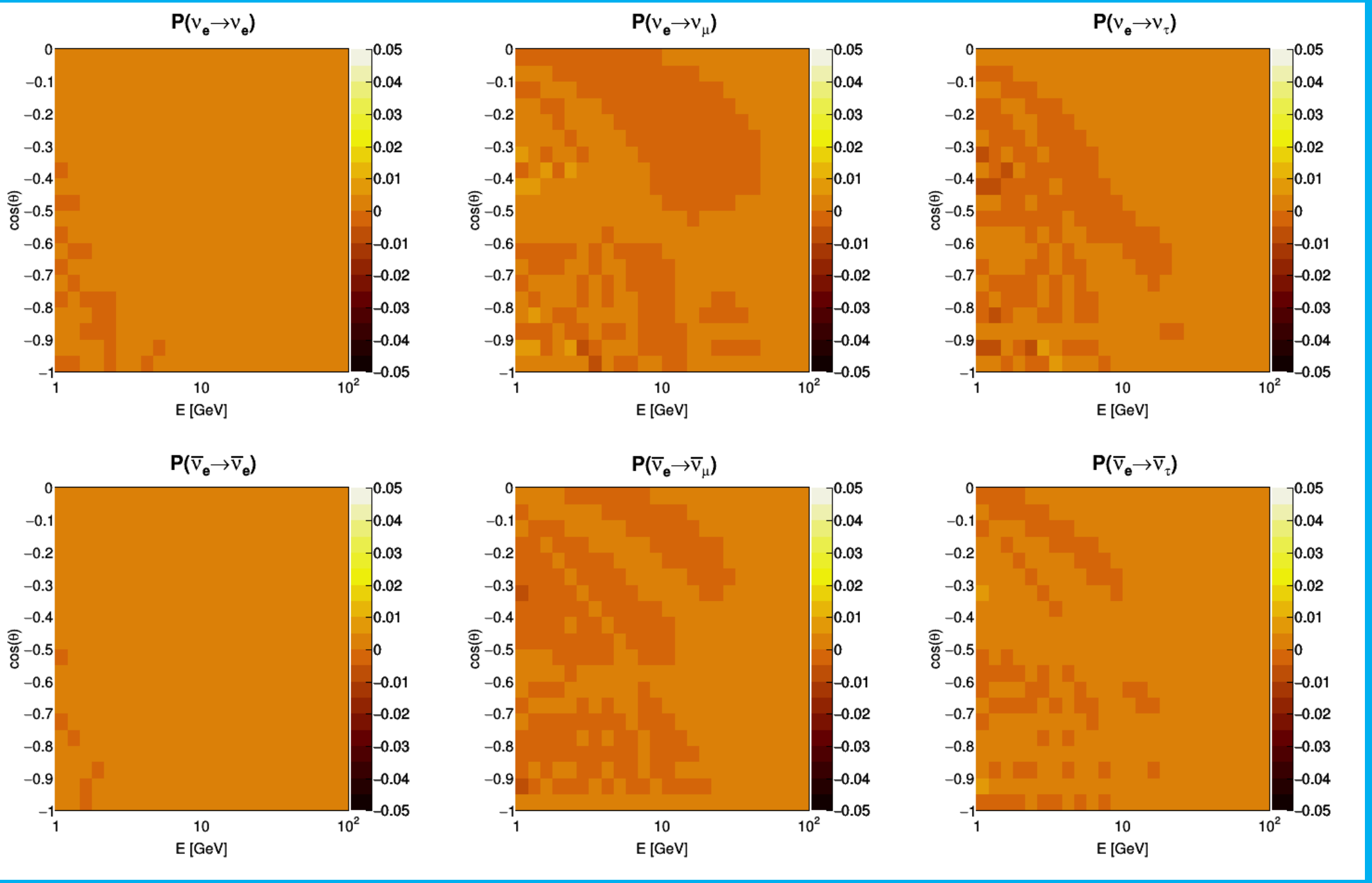
OscProb
Get oscillation probability... OK
200,000 μs elapsed

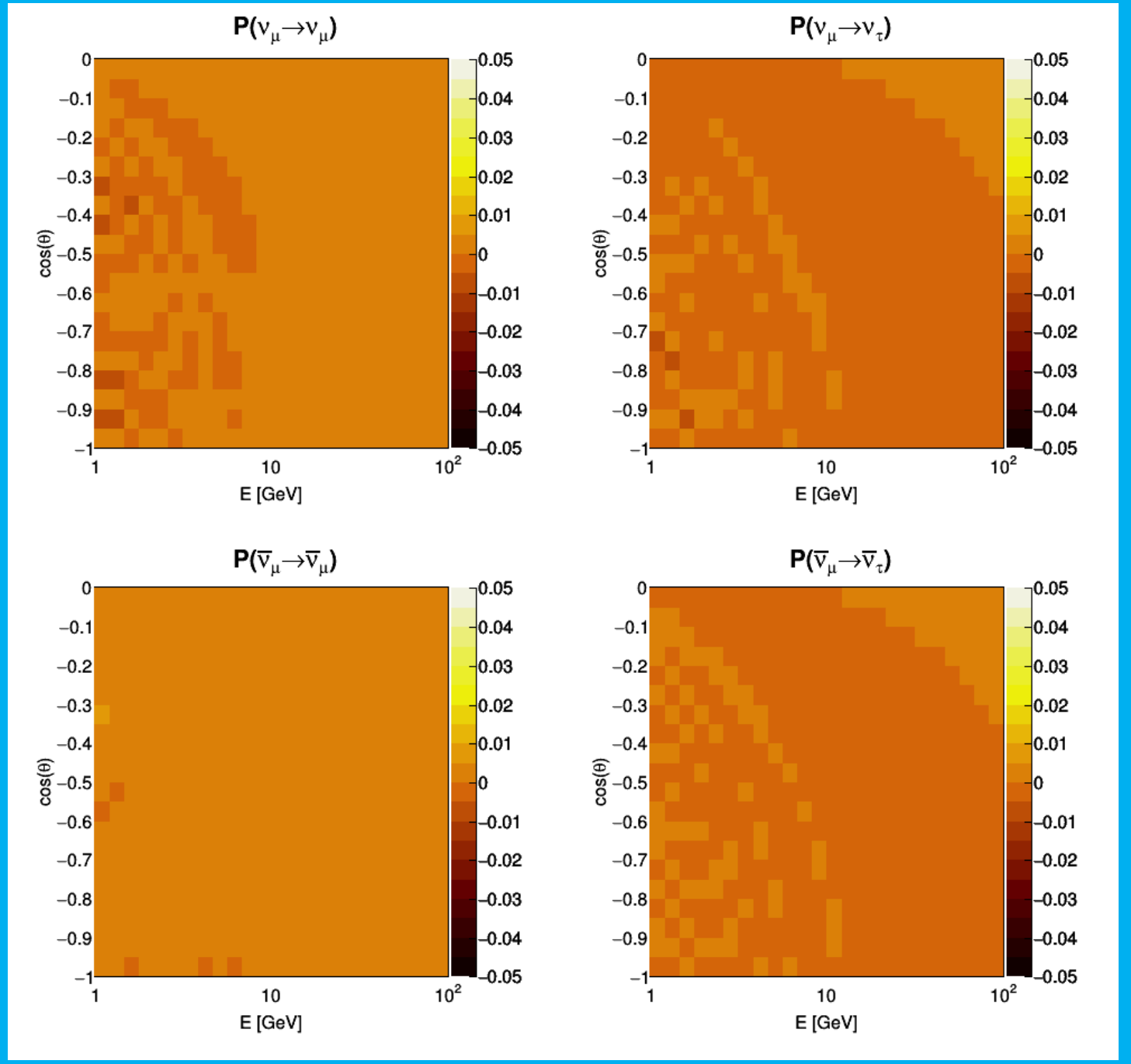
Jpp
Get 6D-interpolation... OK
950 μs elapsed

Jpp
Get 2D-function... OK
750 μs elapsed

Jpp
Get 2D-interpolation... OK
60 μs elapsed

Random point in neutrino parameter space


$P(\nu_e \to \nu_e)$
$P(\nu_e \to \nu_\mu)$
$P(\nu_e \to \nu_\tau)$
$P(\overline{\nu}_e \to \overline{\nu}_e)$
$P(\overline{\nu}_e \to \overline{\nu}_\mu)$
$P(\overline{\nu}_e \to \overline{\nu}_\tau)$
cos(θ)
E [GeV]
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9
−1
1
10
10^2
0.05
0.04
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
−0.04
−0.05


P(ν_μ→ν_μ)
P(ν_μ→ν_τ)
P(ν̄_μ→ν̄_μ)
P(ν̄_μ→ν̄_τ)
cos(θ)
E [GeV]
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9
−1
1
10
10^2
0.05
0.04
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
−0.04
−0.05

Summary & Outlook

- Jpp can readily be used to interpolate neutrino oscillation probability
  - $10 - 100\ \times$ faster than OscProb
  - accuracy < 0.02

- Number of bins and degree of polynomial to be optimised
  - 6D-table $(7 \times 7 \times 7 \times 7 \times 24 \times 20)$ amounts to about 200 MB
  - $1 \rightarrow 0$ degree polynomial interpolation $\Longrightarrow 2 \times$ faster