FunMC: A functional API for building Markov Chains

FunMC is a TensorFlow/JAX library of utilities for accelerating methodological research into sequential, constant-memory algorithms like Markov Chain Monte Carlo (MCMC) and optimization. It does so by providing:

- MCMC building blocks
- Optimization operators
- Streaming statistics operators
- All with an API with minimal abstractions, favoring composition over configuration and utilization of the base language (Python) over a new DSL.

**Core Abstraction: Transition Kernel**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>State</th>
<th>New State</th>
</tr>
</thead>
<tbody>
<tr>
<td>loss_fn</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>learning_rate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HMC step size adaptation**

Optimization can be combined with MCMC to produce adaptive MCMC algorithms.

```python
def transition_kernel(hmc_state, key):
    hmc_key = jax.random.split(key)
    hmc_state, hmc_extra = fun_mc.hamiltonian_monte_carlo(hmc_state, target_log_prob, step_size, num_integrator_steps, seed=hmc_key)
    w, logits = hmc_state.state_extra
    return (hmc_state, key), (w, logits, hmc_extra.is_accepted)

... (w chain, logit chain, is accepted chain) = fun_mc.trace
    (fun_mc.hamiltonian_monte_carlo_init(reparam_w_init, reparam_potential_fn), jax.random.PRNGKey(0)), transition_kernel, num_steps)
```

**Streaming Statistics and Diagnostics**

Streaming statistics enable analysis of an MCMC chain without materializing it.

```python
def transition_kernel(hmc_state, cov_state, rhat_state, key):
    hmc_key = jax.random.split(key)
    hmc_state, hmc_extra = fun_mc.hamiltonian_monte_carlo(hmc_state, target, step_size, num_integrator_steps, seed=hmc_key)
    w, logits = hmc_state.state, hmc_state.state_extra
    cov_state = fun_mc.running_covariance_step(cov_state, (w, logits), axis=0)
    rhat_state = fun_mc.potential_scale_reduction_step(rhat_state, w)
    return (hmc_state, cov_state, rhat_state, key), ()

... (fin cov_state, fin mean accept state, fin rhat state, ), __ = fun_mc.trace
    (fun_mc.hamiltonian_monte_carlo_init(w_init, target),
     fun_mc.running_covariance_init(w_init.shape[-1], y_shape, (np.float32, 2)),
     fun_mc.potential_scale_reduction_init(w_init.shape, np.float32),
     jax.random.PRNGKey(0)), transition_kernel, num_steps)
```

**Markov Chain thinning**

FunMC.trace is itself a transition kernel, enabling chunked computation that can be used for thinning.

```python
... (w chain, logit chain, is accepted chain) = fun_mc.trace
    (fun_mc.hamiltonian_monte_carlo_init(w_init, target), jax.random.PRNGKey(0)),
     lambda *state: fun_mc.trace(state, transition_kernel, num_substeps, trace_mask=False), num_steps // num_substeps)
```